Distributed by:

JAMECO® ELECTRONICS

www.Jameco.com ✦ 1-800-831-4242

The content and copyrights of the attached material are the property of its owner.

Jameco Part Number 35625

Mostek

Z80 PROCESSOR
Technical Manual

MK3880
Central Processing Unit

1979 MICROCOMPUTER
COMPONENT DATA BOOK

(Page intentionally left blank)

MOSTEK®

Z80 MICROCOMPUTER DEVICES

Technical Manual

Z80 Device Family

MK 3880 CENTRAL PROCESSING UNIT

(Page intentionally left blank)

TABLE OF CONTENTS

Chapter		Page
1.0	Introduction	5
2.0	Z80-CPU Architecture	7
3.0	Z80-CPU Pin Description	11
4.0	CPU Timing	15
5.0	Z80-CPU Instruction Set	23
6.0	Flags	43
7.0	Summary of OP Codes and Execution Times	47
8.0	Interrupt Response	59
9.0	Hardware Implementation Examples	65
10.0	Software Implementation Examples	71
11.0	Electrical Specifications	77
12.0	Z80 Instruction Breakdown by Machine Cycle	83
13.0	Package Description and Ordering Information	90

(Page intentionally left blank)

1.0 INTRODUCTION

The term "microcomputer" has been used to describe virtually every type of small computing device designed within the last few years. This term has been applied to everything from simple "microprogrammed" controllers constructed out of TTL MSI up to low end minicomputers with a portion of the CPU constructed out of TTL LSI "bit slices." However, the major impact of the LSI technology within the last few years has been with MOS LSI. With this technology, it is possible to fabricate complete and very powerful computer systems with only a few MOS LSI components.

The Mostek Z80 family of components is a significant advancement in the state-of-art of microcomputers. These components can be configured with any type of standard semiconductor memory to generate computer systems with an extremely wide range of capabilities. For example, as few as two LSI circuits and three standard TTL MSI packages can be combined to form a simple controller. With additional memory and I/O devices a computer can be constructed with capabilities that only a minicomputer could previously deliver. This wide range of computational power allows standard modules to be constructed by a user that can satisfy the requirements of an extremely wide range of applications.

The major reason for MOS LSI domination of the microcomputer market is the low cost of these few LSI components. For example, MOS LSI microcomputers have already replaced TTL logic in such applications as terminal controllers, peripheral device controllers, traffic signal controllers, point of sale terminals, intelligent terminals and test systems. In fact the MOS LSI microcomputer is finding its way into almost every product that now uses electronics and it is even replacing many mechanical systems such as weight scales and automobile controls.

The MOS LSI microcomputer market is already well established and new products using them are being developed at an extraordinary rate. The Mostek Z80 component set has been designed to fit into this market through the following factors:

1. The Z80 is fully software compatible with the popular 8080A CPU offered from several sources. Existing designs can be easily converted to include the Z80 as a superior alternative.

2. The Z80 component set is superior in both software and hardware capabilities to any other 8-bit microcomputer system on the market. These capabilities provide the user with significantly lower hardware and software development costs while also allowing him to offer additional features in his system.

3. A complete development and OEM system product line including full software support is available to enable the user to easily develop new products.

Microcomputer systems are extremely simple to construct using Z80 components. Any such system consists of three parts:

1. CPU (Central Processing Unit)

2. Memory

3. Interface circuits to peripheral devices

The CPU is the heart of the system. Its function is to obtain instructions from the memory and perform the desired operations. The memory is used to contain instructions and in most cases data that is to be processed. For example, a typical instruction sequence may be to read data from a specific peripheral device, store it in a location in memory, check the parity and write it out to another peripheral device. Note that the Mostek component set includes the CPU and various general purpose I/O device controllers, as well as a wide range of memory devices. Thus, all required components can be connected together in a very simple manner with virtually no other external logic. The user's effort then becomes primarily one of software development. That is, the user can concentrate on describing his problem and translating it into a series of instructions that can be loaded into the microcomputer memory. Mostek is dedicated to making this step of software generation as simple as possible. A good example of this is our assembly language in which a simple mnemonic is used to represent every instruction that the CPU can perform. This language is self documenting in such a way that from the mnemonic the user can understand exactly what the instruction is doing without constantly checking back to a complex cross listing.

2.0 Z80-CPU ARCHITECHURE

A block diagram of the internal architecture of the Z80-CPU is shown in Figure 2.0-1 The diagram shows all of the major elements in the CPU and it should be referred to throughout the following description.

Z80-CPU BLOCK DIAGRAM

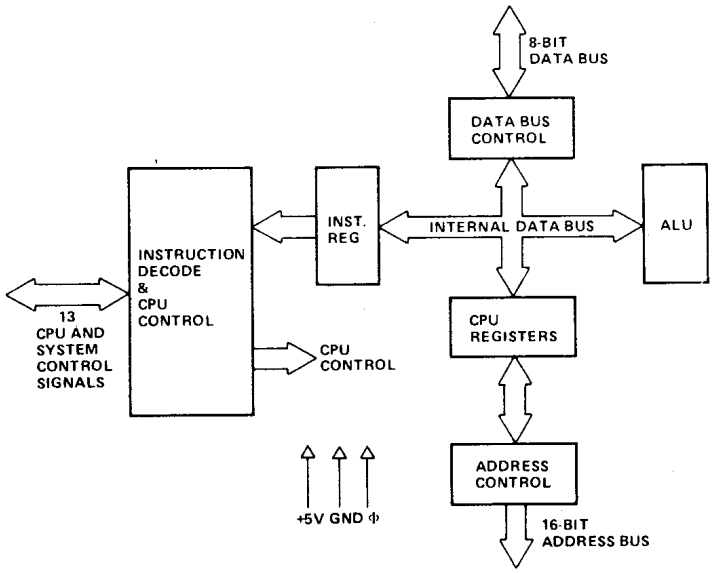


FIGURE 2.0-1

2.1 CPU REGISTERS

The Z80-CPU contains 208 bits of R/W memory that are accessible to the programmer. Figure 2.0-2 illustrates how this memory is configured into eighteen 8-bit registers and four 16-bit registers. All Z80 registers are implemented using static RAM. The registers include two sets of six general purpose registers that may be used individually as 8-bit registers or in pairs as 16-bit registers. There are also two sets of accumulator and flag registers.

Special Purpose Registers

1. **Program Counter (PC).** The program counter holds the 16-bit address of the current instruction being fetched from memory. The PC is automatically incremented after its contents have been transferred to the address lines. When a program jump occurs the new value is automatically placed in the PC, overriding the incrementer.

2. **Stack Pointer (SP).** The stack pointer holds the 16-bit address of the current top of a stack located anywhere in external system RAM memory. The external stack memory is organized as a last-in first-out (LIFO) file. Data can be pushed onto the stack from specific CPU registers or popped off of the stack into specific CPU registers through the execution of PUSH and POP instructions. The data popped from the stack is always the last data pushed onto it. The stack allows simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data manipulation.

Z80-CPU REGISTER CONFIGURATION

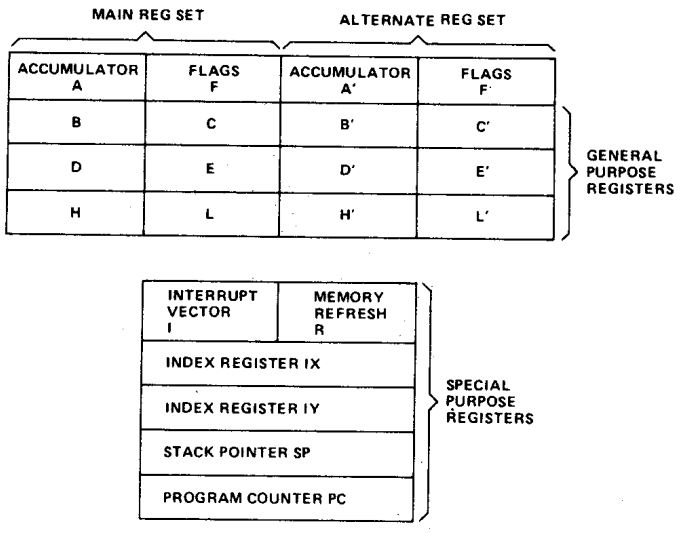


FIGURE 2.0-2

3. **Two Index Registers (IX & IY).** The two independent index registers hold a 16-bit base address that is used in indexed addressing modes. In this mode, an index register is used as a base to point to a region in memory from which data is to be stored or retrieved. An additional byte is included in indexed instructions to specify a displacement from this base. This displacement is specified as a two's complement signed integer. This mode of addressing greatly simplifies many types of programs, especially where tables of data are used.

4. **Interrupt Page Address Register (I).** The Z80-CPU can be operated in a mode where an indirect call to any memory location can be achieved in response to an interrupt. The I Register is used for this purpose to store the high order 8-bits of the indirect address while the interrupting device provides the lower 8-bits of the address. This feature allows interrupt routines to be dynamically located anywhere in memory with absolute minimal access time to the routine.

5. **Memory Refresh Register (R).** The Z80-CPU contains a memory refresh counter to enable dynamic memories to be used with the same ease as static memories. This 7-bit register is automatically incremented after each instruction fetch. The data in the refresh counter is sent out on the lower portion of the address bus along with a refresh control signal while the CPU is decoding and executing the fetched instruction. This mode of refresh is totally transparent to the programmer and does not slow down the CPU operation. The programmer can load the R register for testing purposes, but this register is normally not used by the programmer.

Accumulator and Flag Registers

The CPU includes two independent 8-bit accumulators and associated 8-bit flag registers. The accumulator holds the results of 8-bit arithmetic or logical operations while the flag register indicates specific conditions for 8 or 16-bit operations, such as indicating whether or not the result of an operation is equal to zero. The programmer selects the accumulator and flag pair that he wishes to work with with a single exchange instruction so that he may easily work with either pair.

General Purpose Registers

There are two matched sets of general purpose registers, each set containing six 8-bit registers that may be used individually as 8-bit registers or as 16-bit register pairs by the programmer. One set is called BC, DE, and HL while the complementary set is called BD', DE' and HL'. At any one time the programmer can select either set of registers to work with through a single exchange command for the entire set. In systems where fast interrupt response is required, one set of general purpose registers and an accumulator/flag register may be reserved for handling this very fast routine. Only a simple exchange command need be executed to go between the routines. This greatly reduces interrupt service time by eliminating the requirement for saving and retrieving register contents in the external stack during interrupt or subroutine processing. These general purpose registers are used for a wide range of applications by the programmer. They also simplify programming, especially in ROM based systems where little external read/write memory is available.

2.2 ARITHMETIC & LOGIC UNIT (ALU)

The 8-bit arithmetic and logical instructions of the CPU are executed in the ALU. Internally the ALU communicates with the registers and the external data bus on the internal data bus. The type of functions performed by the ALU include:

Add	Left or right shifts or rotates (arithmetic and logical)
Subtract	Increment
Logical AND	Decrement
Logical OR	Set bit
Logical Exclusive OR	Reset bit
Compare	Test bit

2.3 INSTRUCTION REGISTER AND CPU CONTROL

As each instruction is fetched from memory, it is placed in the instruction register and decoded. The control section performs this function and then generates and supplies all of the control signals necessary to read or write data from or to the registers, controls the ALU and provides all required external control signals.

(Page intentionally left blank)

3.0 Z80-CPU PIN DESCRIPTION

The Z80-CPU is packaged in an industry standard 40 pin Dual In-Line Package. The I/O pins are shown in Figure 3.0-1 and the function of each is described below.

Z80 PIN CONFIGURATION

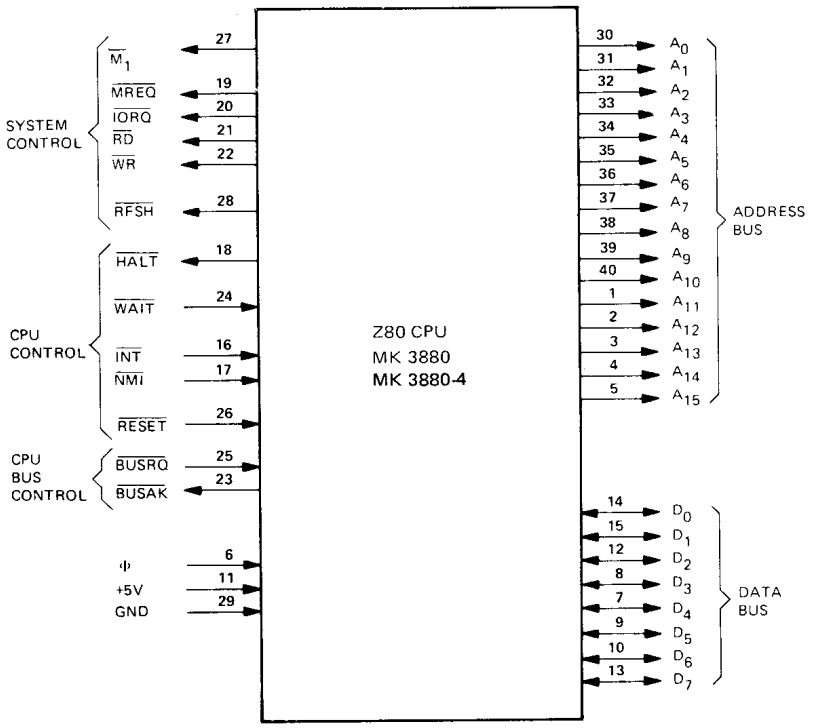


FIGURE 3.0-1

A_0-A_{15} (Address Bus)

Tri-state output, active high. A_0-A_{15} constitute a 16-bit address bus. The address bus provides the address for memory (up to 64K bytes) data exchanges and for I/O device data exchanges. I/O addressing uses the 8 lower address bits to allow the user to directly select up to 256 input or 256 output ports. A_0 is the least significant address bit. During refresh time, the lower 7 bits contain a valid refresh address.

D_0-D_7 (Data Bus)

Tri-state input/output, active high. D_0-D_7 constitute an 8-bit bidirectional data bus. The data bus is used for data exchanges with memory and I/O devices.

$\overline{M1}$ (Machine Cycle one)

Output, active low. $\overline{M1}$ indicates that the current machine cycle is the OP code fetch cycle of an instruction execution. Note that during execution of 2-byte op-codes, $\overline{M1}$ is generated as each op code byte is fetched. These two byte op-codes always begin with CBH, DDH, EDH, or FDH. $\overline{M1}$ also occurs with $\overline{IORQ}$ to indicate an interrupt acknowledge cycle.

$\overline{MREQ}$ (Memory Request)

Tri-state output, active low. The memory request signal indicates that the address bus holds a valid address for a memory read or memory write operation.

$\overline{IORQ}$
(Input/Output Request)

Tri-state output, active low. The $\overline{IORQ}$ signal indicates that the lower half of the address bus holds a valid I/O address for a I/O read or write operation. An $\overline{IORQ}$ signal is also generated with an $\overline{M_1}$ signal when an interrupt is being acknowledged to indicate that an interrupt response vector can be placed on the data bus. Interrupt Acknowledge operations occur during M_1 time while I/O operations never occur during M_1 time.

$\overline{RD}$
(Memory Read)

Tri-state output, active low. $\overline{RD}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.

$\overline{WR}$
(Memory Write)

Tri-state output, active low. $\overline{WR}$ indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.

$\overline{RFSH}$
(Refresh)

Output, active low. $\overline{RFSH}$ indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and current $\overline{MREQ}$ signal should be used to do a refresh read to all dynamic memories. A_7 is a logic zero and the upper 8 bits of the Address Bus contains the I Register.

$\overline{HALT}$
(Halt state)

Output, active low. $\overline{HALT}$ indicates that the CPU has executed a HALT software instruction and is awaiting either a non maskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU executes NOP's to maintain memory refresh activity.

$\overline{WAIT}$*
(Wait)

Input, active low. $\overline{WAIT}$ indicates to the Z80-CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter wait states for as long as this signal is active. This signal allows memory or I/O devices of any speed to be synchronized to the CPU.

$\overline{INT}$
(Interrupt Request)

Input, active low. The Interrupt Request signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal software controlled interrupt enable flip-flop (IFF) is enabled and if the $\overline{BUSRQ}$ signal is not active. When the CPU accepts the interrupt, an acknowledge signal ($\overline{IORQ}$ during M_1 time) is sent out at the beginning of the next instruction cycle. The CPU can respond to an interrupt in three different modes that are described in detail in section 8.

$\overline{NMI}$

Input, negative edge triggered. The non maskable interrupt request line has a higher priority than $\overline{INT}$ and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. $\overline{NMI}$ automatically forces the Z80-CPU to restart to location 0066_H. The program counter is automatically saved in the external stack so that the user can return to the program that was interrupted. Note that continuous WAIT cycles can prevent the current instruction from ending, and that a $\overline{BUSRQ}$ will override a $\overline{NMI}$.

$\overline{\text{RESET}}$	Input, active low. $\overline{\text{RESET}}$ forces the program counter to zero and initializes the CPU. The CPU initialization includes: 1) Disable the interrupt enable flip-flop 2) Set Register I = 00_H 3) Set Register R = 00_H 4) Set Interrupt Mode 0 During reset time, the address bus and data bus go to a high impedance state and all control output signals go to the inactive state. No refresh occurs.
$\overline{\text{BUSRQ}}$ (Bus Request)	Input, active low. The bus request signal is used to request the CPU address bus, data bus and tri-state output control signals to go to a high impedance state so that other devices can control these buses. When $\overline{\text{BUSRQ}}$ is activated, the CPU will set these buses to a high impedance state as soon as the current CPU machine cycle is terminated.
$\overline{\text{BUSAK}}$* (Bus Acknowledge)	Output, active low. Bus acknowledge is used to indicate to the requesting device that the CPU address bus, data bus and tri-state control bus signals have been set to their high impedance state and the external device can now control these signals.
Φ	Single phase system clock.

*While the Z80-CPU is in either a $\overline{\text{WAIT}}$ state or a Bus Acknowledge condition, Dynamic Memory Refresh will not occur.

(Page intentionally left blank)

4.0 CPU TIMING

The Z80-CPU executes instructions by stepping through a very precise set of a few basic operations. These include:

Memory read or write

I/O device read or write

Interrupt acknowledge

All instructions are merely a series of these basic operations. Each of these basic operations can take from three to six clock periods to complete or they can be lengthened to synchronize the CPU to the speed of external devices. The basic clock periods are referred to as T states and the basic operations are referred to as M (for machine) cycles. Figure 4.0-0 illustrates how a typical instruction will be merely a series of specific M and T cycles. Notice that this instruction consists of three machine cycles (M1, M2 and M3). The first machine cycle of any instruction is a fetch cycle which is four, five or six T states long (unless lengthened by the wait signal which will be fully described in the next section). The fetch cycle (M1) is used to fetch the OP code of the next instruction to be executed. Subsequent machine cycles move data between the CPU and memory or I/O devices and they may have anywhere from three to five T cycles (again they may be lengthened by wait states to synchronize the external devices to the CPU). The following paragraphs describe the timing which occurs within any of the basic machine cycles. In section 7, the exact timing for each instruction is specified.

BASIC CPU TIMING EXAMPLE

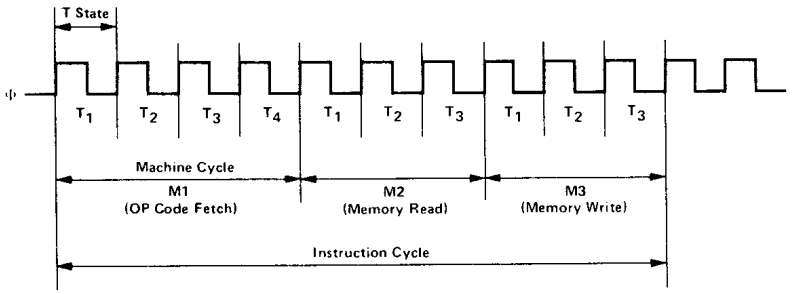


FIGURE 4.0-0

All CPU timing can be broken down into a few very simple timing diagrams as shown in Figure 4.0-1 through 4.0-7. These diagrams show the following basic operations with and without wait states (wait states are added to synchronize the CPU to slow memory or I/O devices).

4.0-1. Instruction OP code fetch (M1 cycle)
4.0-2. Memory data read or write cycles
4.0-3. I/O read or write cycles
4.0-4. Bus Request/Acknowledge Cycle
4.0-5. Interrupt Request/Acknowledge Cycle
4.0-6. Non maskable Interrupt Request/Acknowledge Cycle
4.0-7. Exit from a HALT instruction

INSTRUCTION FETCH

Figure 4.0-1 shows the timing during an M1 cycle (OP code fetch). Notice that the PC is placed on the address bus at the beginning of the M1 cycle. One half clock time later the $\overline{MREQ}$ signal goes active. At this time the address to the memory has had time to stabilize so that the falling edge of $\overline{MREQ}$ can be used directly as a chip enable clock to dynamic memories. The $\overline{RD}$ line also goes active to indicate that the memory read data should be enabled onto the CPU data bus. The CPU samples the data from the memory on the data bus with the rising edge of the clock of state T3 and this same edge is used by the CPU to turn off the $\overline{RD}$ and $\overline{MREQ}$ signals. Thus the data has already been sampled by the CPU before the $\overline{RD}$ signal becomes inactive. Clock state T3 and T4 of a fetch cycle are used to refresh dynamic memories. (The CPU uses this time to decode and execute the fetched instruction so that no other operation could be performed at this time). During T3 and T4 the lower 7 bits of the address bus contain a memory refresh address and the $\overline{RFSH}$ signal becomes active to indicate that a refresh read of all dynamic memories should be accomplished. Notice that a $\overline{RD}$ signal is not generated during refresh time to prevent data from different memory segments from being gated onto the data bus. The $\overline{MREQ}$ signal during refresh time should be used to perform a refresh read of all memory elements. The refresh signal can not be used by itself since the refresh address is only guaranteed to be stable during $\overline{MREQ}$ time.

Z80 Device Family

INSTRUCTION OP CODE FETCH

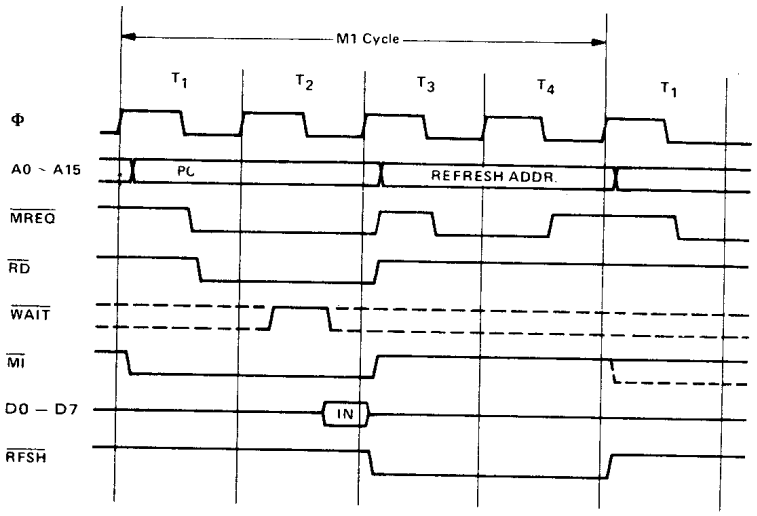


FIGURE 4.0-1

Figure 4.0-1A illustrates how the fetch cycle is delayed if the memory activates the $\overline{WAIT}$ line. During T2 and every subsequent Tw, the CPU samples the $\overline{WAIT}$ line with the falling edge of Φ. If the $\overline{WAIT}$ line is active at this time, another wait state will be entered during the following cycle. Using this technique the read cycle can be lengthened to match the access time of any type of memory device.

INSTRUCTION OP CODE FETCH WITH WAIT STATES

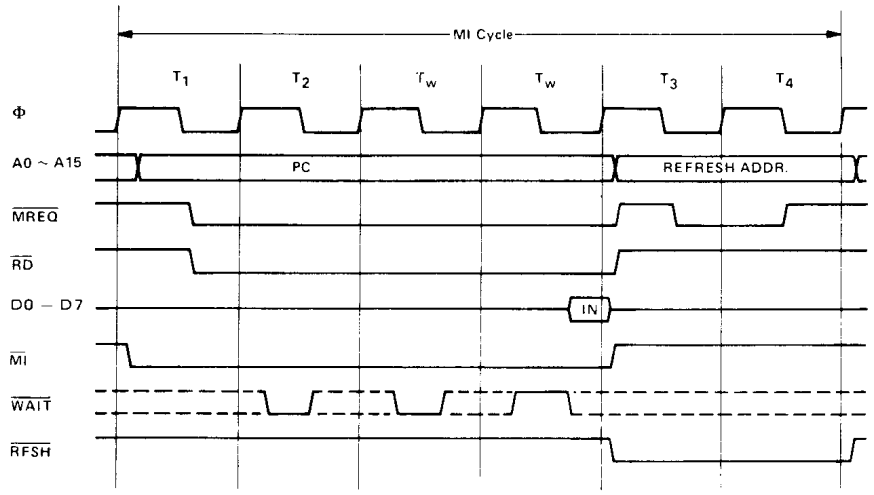


FIGURE 4.0-1A

MEMORY READ OR WRITE

Figure 4.0-2 illustrates the timing of memory read or write cycles other than an OP code fetch (M1 cycle). These cycles are generally three clock periods long unless wait states are requested by the memory via the WAIT signal. The MREQ signal and the RD signal are used the same as in the fetch cycle. In the case of a memory write cycle, the MREQ also becomes active when the address bus is stable so that it can be used directly as a chip enable for dynamic memories. The WR line is active when data on the data bus is stable so that it can be used directly as a R/W pulse to virtually any type of semiconductor memory. Furthermore the WR signal goes inactive one half T state before the address and data bus contents are changed so that the overlap requirements for virtually any type of semiconductor memory type will be met.

MEMORY READ OR WRITE CYCLES

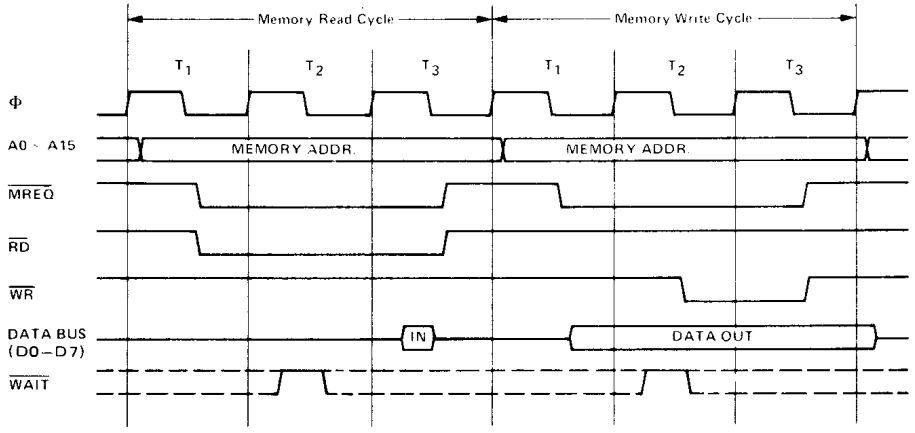


FIGURE 4.0-2

Figure 4.0-2A illustrates how a $\overline{WAIT}$ request signal will lengthen any memory read or write operation. This operation is identical to that previously described for a fetch cycle. Notice in this figure that a separate read and a separate write cycle are shown in the same figure although read and write cycles can never occur simultaneously.

MEMORY READ OR WRITE CYCLES WITH WAIT STATES

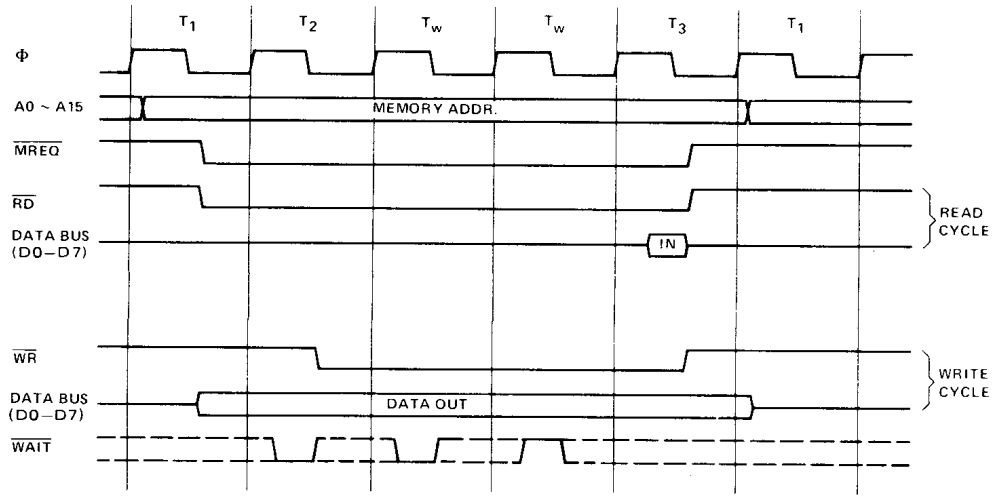


FIGURE 4.0-2A

INPUT OR OUTPUT CYCLES

Figure 4.0-3 illustrates an I/O read or I/O write operation. Notice that during I/O operations a single wait state is automatically inserted. The reason for this is that during I/O operations, the time from when the $\overline{IORQ}$ signal goes active until the CPU must sample the $\overline{WAIT}$ line is very short and without this extra state sufficient time does not exist for an I/O port to decode its address and activate the $\overline{WAIT}$ line if a wait is required. Also, without this wait state it is difficult to design MOS I/O devices that can operate at full CPU speed. During this wait state time the $\overline{WAIT}$ request signal is sampled. During a read I/O operation, the $\overline{RD}$ line is used to enable the addressed port onto the data bus just as in the case of a memory read. For I/O write operations, the $\overline{WR}$ line is used as a clock to the I/O port, again with sufficient overlap timing automatically provided so that the rising edge may be used as a data clock.

Figure 4.0-3A illustrates how additional wait states may be added with the $\overline{WAIT}$ line. The operation is identical to that previously described.

BUS REQUEST/ACKNOWLEDGE CYCLE

Figure 4.0-4 illustrates the timing for a Bus Request/Acknowledge cycle. The $\overline{BUSRQ}$ signal is sampled by the CPU with the rising edge of the last clock period of any machine cycle. If the $\overline{BUSRQ}$ signal is active, the CPU will set its address, data and tri-state control signals to the high impedance state with the rising edge of the next clock pulse. At that time any external device can control the buses to transfer data between memory and I/O devices. (This is generally known as Direct Memory Access [DMA] using cycle stealing). The maximum time for the CPU to respond to a bus request is the length of a machine cycle and the external controller can maintain control of the bus for as many clock cycles as is desired. Note, however, that if very long DMA cycles are used, and dynamic memories are being used, the external controller must also perform the refresh function. This situation only occurs if very large blocks of data are transferred under DMA control. Also note that during a bus request cycle, the CPU cannot be interrupted by either a $\overline{NMI}$ or an $\overline{INT}$ signal.

INPUT OR OUTPUT CYCLES

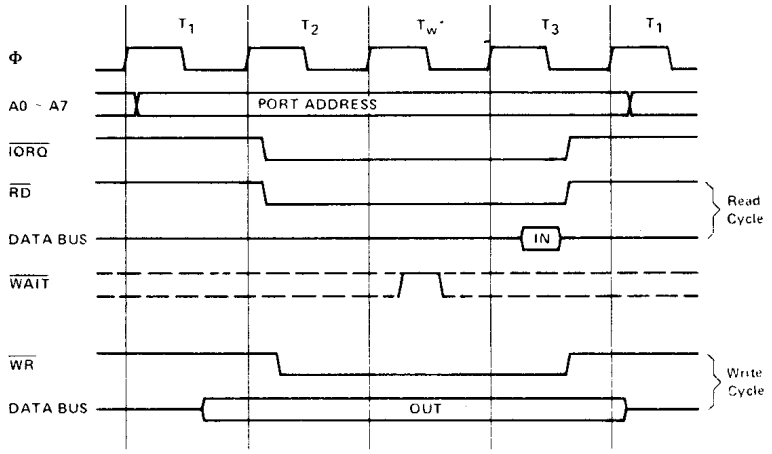


FIGURE 4.0-3

INPUT OR OUTPUT CYCLES WITH WAIT STATES

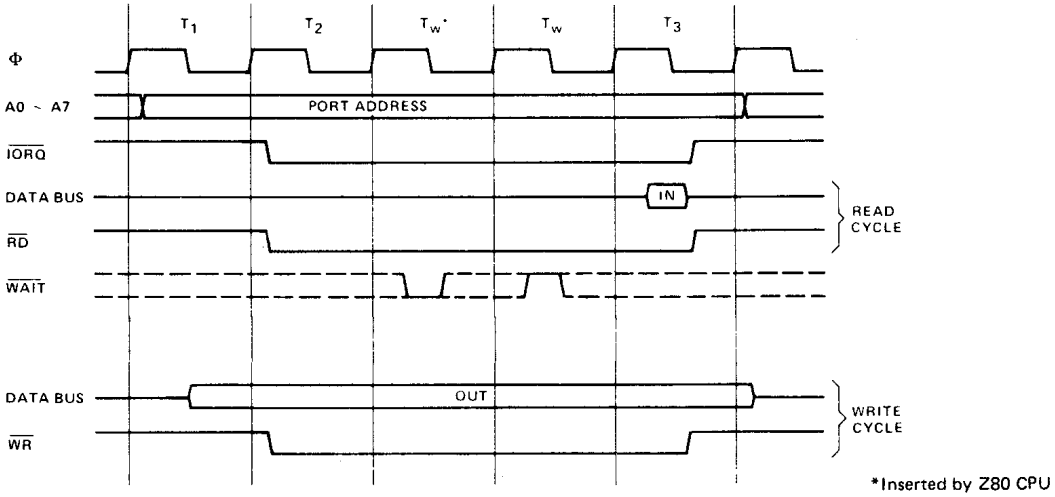


FIGURE 4.0-3A

BUS REQUEST/ACKNOWLEDGE CYCLE

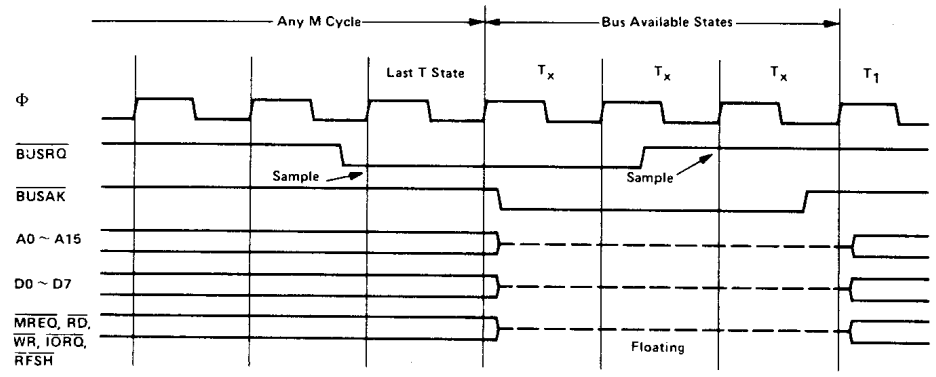
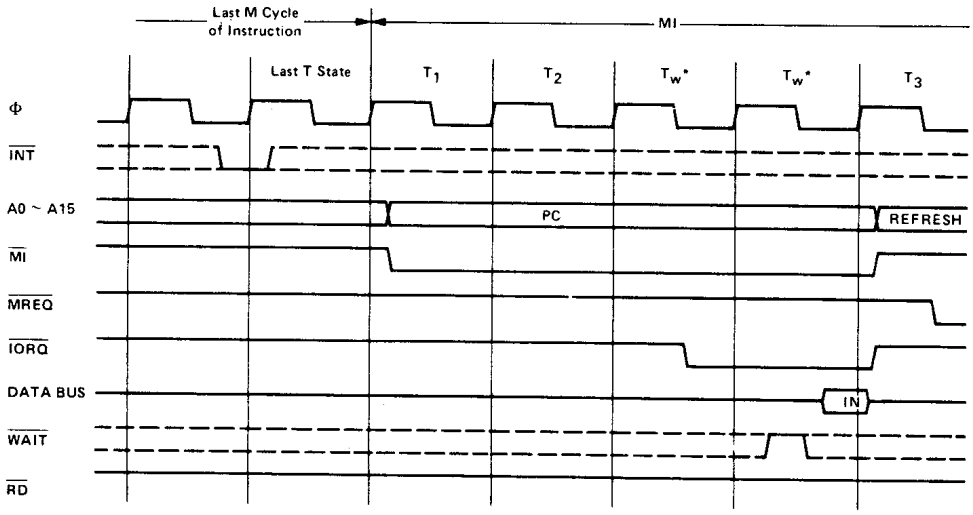


FIGURE 4.0-4

INTERRUPT REQUEST/ ACKNOWLEDGE CYCLE

Figure 4.0-5 illustrates the timing associated with an interrupt cycle. The interrupt signal ($\overline{INT}$) is sampled by the CPU with the rising edge of the last clock at the end of any instruction. The signal will not be accepted if the internal CPU software controlled interrupt enable flip-flop is not set or if the $\overline{BUSRQ}$ signal is active. When the signal is accepted a special M1 cycle is generated. During this special M1 cycle the $\overline{IORQ}$ signal becomes active (instead of the normal $\overline{MREQ}$) to indicate that the interrupting device can place an 8-bit vector on the data bus. Notice that two wait states are automatically added to this cycle. These states are added so that a ripple priority interrupt scheme can be easily implemented. The two wait states allow sufficient time for the ripple signals to stablilize and identify which I/O device must insert the response vector. Refer to section 8.0 for details on how the interrupt response vector is utilized by the CPU.

INTERRUPT REQUEST/ACKNOWLEDGE CYCLE

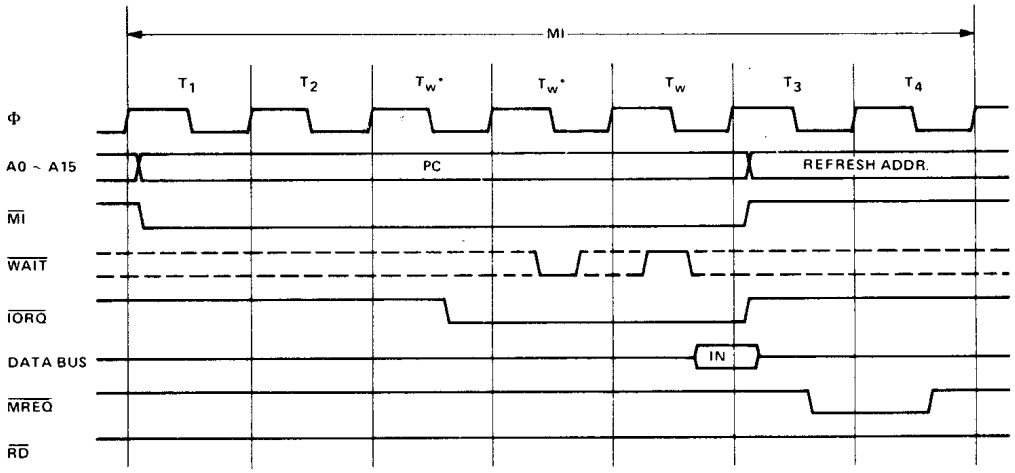


Mode **0** shown

FIGURE 4.0-5

Figure 4.0-5A illustrates how additional wait states can be added to the interrupt response cycle. Again the operation is identical to that previously described.

INTERRUPT REQUEST/ACKNOWLEDGE WITH WAIT STATES



Mode 0 shown

FIGURE 4.0-5A

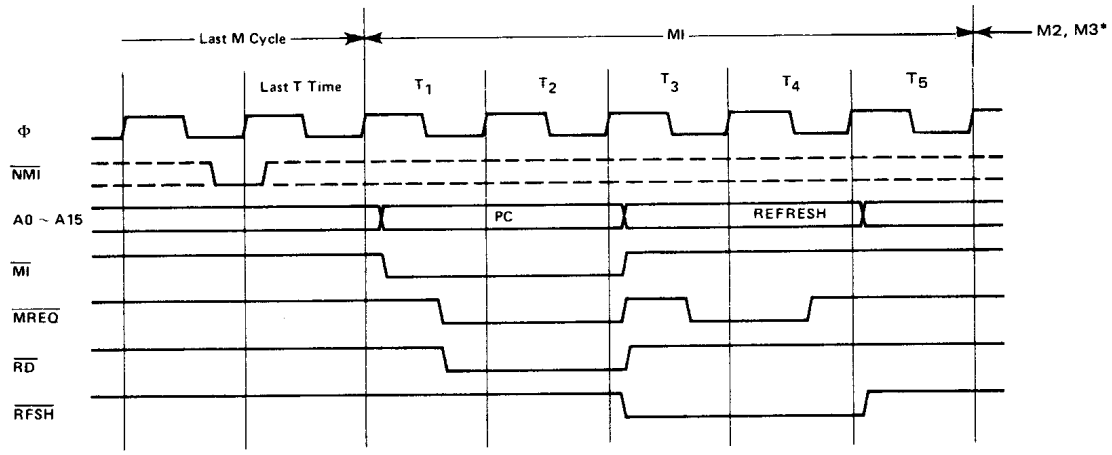
NON MASKABLE INTERRUPT RESPONSE

Figure 4.0-6 illustrates the request/acknowledge cycle for the non-maskable interrupt. A pulse on the $\overline{NMI}$ input sets an internal NMI latch which is tested by the CPU at the end of every instruction. This NMI latch is sampled at the same time as the interrupt line, but this line has priority over the normal interrupt and it can not be disabled under software control. Its usual function is to provide immediate response to important signals such as an impending power failure. The CPU response to a non maskable interrupt is similar to a normal memory read operation. The only difference being that the content of the data bus is ignored while the processor automatically stores the PC in the external stack and jumps to location 0066_H. The service routine for the non maskable interrupt must begin at this location if this interrupt is used.

HALT EXIT

Whenever a software halt instruction is executed the CPU begins executing NOP's until an interrupt is received (either a non-maskable or a maskable interrupt while the interrupt flip flop is enabled). The two interrupt lines are sampled with the rising clock edge during each T4 state as shown in Figure 4.0-7. If a non-maskable interrupt has been received or a maskable interrupt has been received and the interrupt enable flip-flop is set, then the halt state will be exited on the next rising clock edge. The following cycle will then be an interrupt acknowledge cycle corresponding to the type of interrupt that was received. If both are received at this time, then the non maskable one will be acknowledged since it was highest priority. The purpose of executing NOP instructions while in the halt state is to keep the memory refresh signals active. Each cycle in the halt state is a normal M1 (fetch) cycle except that the data received from the memory is ignored and a NOP instruction is forced internally to the CPU. The halt acknowledge signal is active during this time to indicate that the processor is in the halt state.

NON MASKABLE INTERRUPT REQUEST OPERATION



*M2 and M3 are stack write operations

FIGURE 4.0-6

HALT EXIT

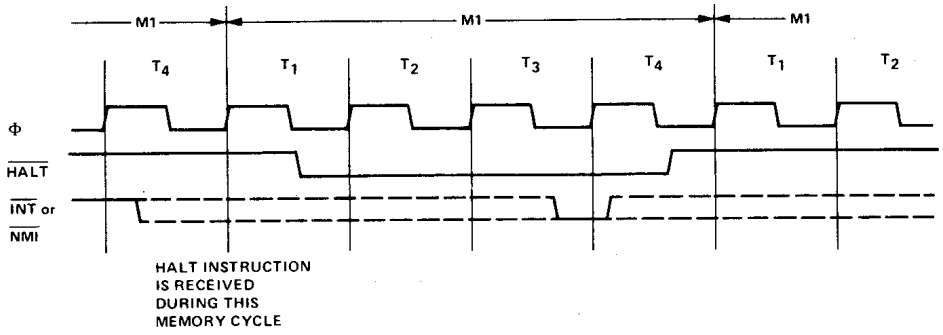


FIGURE 4.0-7

5.0 Z80-CPU INSTRUCTION SET

The Z80-CPU can execute 158 different instruction types including all 78 of the 8080A CPU. The instructions can be broken down into the following major groups:

- Load and Exchange
- Block Transfer and Search
- Arithmetic and Logical
- Rotate and Shift
- Bit Manipulation (set, reset, test)
- Jump, Call and Return
- Input/Output
- Basic CPU Control

5.1 INTRODUCTION TO INSTRUCTION TYPES

The load instructions move data internally between CPU registers or between CPU registers and external memory. All of these instructions must specify a source location from which the data is to be moved and a destination location. The source location is not altered by a load instruction. Examples of load group instructions include moves between any of the general purpose registers such as move the data to Register B from Register C. This group also includes load immediate to any CPU register or to any external memory location. Other types of load instructions allow transfer between CPU registers and memory locations. The exchange instructions can trade the contents of two registers.

A unique set of block transfer instructions is provided in the Z80. With a single instruction a block of memory of any size can be moved to any other location in memory. This set of block moves is extremely valuable when large strings of data must be processed. The Z80 block search instructions are also valuable for this type of processing. With a single instruction, a block of external memory of any desired length can be searched for any 8-bit character. Once the character is found the instruction automatically terminates. Both the block transfer and the block search instructions can be interrupted during their execution so as to not occupy the CPU for long periods of time.

The arithmetic and logical instructions operate on data stored in the accumulator and other general purpose CPU registers or external memory locations. The results of the operations are placed in the accumulator and the appropriate flags are set according to the result of the operation. An example of an arithmetic operation is adding the accumulator to the contents of an external memory location. The results of the addition are placed in the accumulator. This group also includes 16-bit addition and subtraction between 16-bit CPU registers.

The bit manipulation instructions allow any bit in the accumulator, any general purpose register or any external memory location to be set, reset or tested with a single instruction. For example, the most significant bit of register H can be reset. This group is especially useful in control applications and for controlling software flags in general purpose programming.

The jump, call and return instructions are used to transfer between various locations in the user's program. This group uses several different techniques for obtaining the new program counter address from specific external memory locations. A unique type of jump is the restart instruction. This instruction actually contains the new address as a part of the 8-bit OP code. This is possible since only 8 separate addresses located in page zero of the external memory may be specified. Program jumps may also be achieved by loading register HL, IX or IY directly into the PC, thus allowing the jump address to be a complex function of the routine being executed.

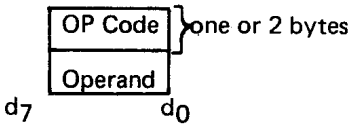
The input/output group of instructions in the Z80 allow for a wide range of transfers between external memory locations or the general purpose CPU registers, and the external I/O devices. In each case, the port number is provided on the lower 8 bits of the address bus during any I/O transaction. One instruction allows this port number to be specified by the second byte of the instruction while other Z80 instructions allow it to be specified as the content of the C register. One major advantage of using the C register as a pointer to the I/O device is that it allows different I/O ports to share common software driver routines. This is not possible when the address is part of the OP code if the routines are stored in ROM. Another feature of these input instructions is that they set the flag register automatically so that additional operations are not required to determine the state of the input data (for example its parity). The Z80-CPU includes single instructions that can move blocks or data (up to 256 bytes) automatically to or from any I/O port directly to any memory location. In conjunction with the dual set of general purpose registers, these instructions provide for fast I/O block transfer rates. The value of this I/O instruction set is demonstrated by the fact that the Z80-CPU can provide all required floppy disk formatting (i.e., the CPU provides the preamble, address, data and enables the CRC codes) on double density floppy disk drives on an interrupt driven basis.

Finally, the basic CPU control instructions allow various options and modes. This group includes instructions such as setting or resetting the interrupt enable flip flop or setting the mode of interrupt response.

5.2 ADDRESSING MODES

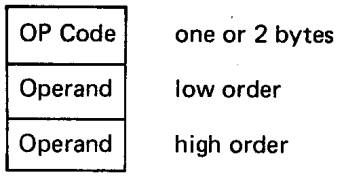
Most of the Z80 instructions operate on data stored in internal CPU registers, external memory or in the I/O ports. Addressing refers to how the address of this data is generated in each instruction. This section gives a brief summary of the types of addressing used in the Z80 while subsequent sections detail the type of addressing available for each instruction group.

Immediate. In this mode of addressing the byte following the OP code in memory contains the actual operand.



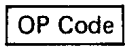
Examples of this type of instruction would be to load the accumulator with a constant, where the constant is the byte immediately following the OP code.

Immediate Extended. This mode is merely an extension of immediate addressing in that the two bytes following the op codes are the operand.



Examples of this type of instruction would be to load the HL register pair (16-bit register) with 16 bits (2 bytes) of data.

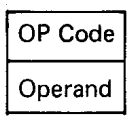
Modified Page Zero Addressing. The Z80 has a special single byte call instruction to any of 8 locations in page zero of memory. This instruction (which is referred to as a restart) sets the PC to an effective address in page zero. The value of this instruction is that it allows a single byte to specify a complete 16-bit address where commonly called subroutines are located, thus saving memory space.



one byte

b_7 b_0 Effective address is (00$b_5b_4b_3$000)

Relative Addressing. Relative addressing uses one byte of data following the OP code to specify a displacement from the existing program to which a program jump can occur. This displacement is a signed two's complement number that is added to the address of the OP code of the following instruction.

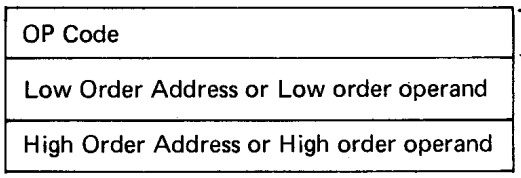


Jump relative (one byte OP code)

8-bit two's complement displacement added to Address (A+2)

The value of relative addressing is that it allows jumps to nearby locations while only requiring two bytes of memory space. For most programs, relative jumps are by far the most prevalent type of jump due to the proximity of related program segments. Thus, these instructions can significantly reduce memory space requirements. The signed displacement can range between +127 and -128 from A + 2. This allows for a total displacement of +129 to -126 from the jump relative OP code address. Another major advantage is that it allows for relocatable code.

Extended Addressing. Extended Addressing provides for two bytes (16 bits) of address to be included in the instruction. This data can be an address to which a program can jump or it can be an address where an operand is located.

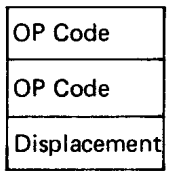


one or two bytes

Extended addressing is required for a program to jump from any location in memory to any other location, or load and store data in any memory location.

When extended addressing is used to specify the source or destination address of an operand, the notation (nn) will be used to indicate the content of memory at nn, where nn is the 16-bit address specified in the instruction. This means that the two bytes of address nn are used as a pointer to a memory location. The use of the parentheses always means that the value enclosed within them is used as a pointer to a memory location. For example, (1200) refers to the contents of memory at location 1200.

Indexed Addressing. In this type of addressing, the byte of data following the OP code contains a displacement which is added to one of the two index registers (the OP code specifies which index register is used) to form a pointer to memory. The contents of the index register are not altered by this operation.



two byte OP code

Operand added to index register to form a pointer to memory.

An example of an indexed instruction would be to load the contents of the memory location (Index Register + Displacement) into the accumulator. The displacement is a signed two's complement number. Indexed addressing greatly simplifies programs using tables of data since the index register can point to the start of any table. Two index registers are provided since very often operations require two or more tables. Indexed addressing also allows for relocatable code.

The two index registers in the Z80 are referred to as IX and IY. To indicate indexed addressing the notation:

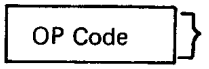
(IX+d) or (IY+d)

is used. here d is the displacement specified after the OP code. The parentheses indicate that this value is used as a pointer to external memory.

Register Addressing. Many of the Z80 OP codes contain bits of information that specify which CPU register is to be used for an operation. An example of register addressing would be to load the data in register B into register C.

Implied Addressing. Implied addressing refers to operations where the OP code automatically implies one or more CPU registers as containing the operands. An example is the set of arithmetic operations where the accumulator is always implied to be the destination of the results.

Register Indirect Addressing. This type of addressing specifies a 16-bit CPU register pair (such as HL) to be used as a pointer to any location in memory. This type of instruction is very powerful and it is used in a wide range of applications.



one or two bytes

An example of this type of instruction would be to load the accumulator with the data in the memory location pointed to by the HL register contents. Indexed addressing is actually a form of register indirect addressing except that a displacement is added with indexed addressing. Register indirect addressing allows for very powerful but simple to implement memory accesses. The block move and search commands in the Z80 are extensions of this type of addressing where automatic register incrementing, decrementing and comparing has been added. The notation for indicating register indirect addressing is to put parentheses around the name of the register that is to be used as the pointer. For example, the symbol

(HL)

specifies that the contents of the HL register are to be used as a pointer to a memory location. Often register indirect addressing is used to specify 16-bit operands. In this case, the register contents point to the lower order portion of the operand while the register contents are automatically incremented to obtain the upper portion of the operand.

Bit Addressing. The Z80 contains a large number of bit set, reset and test instructions. These instructions allow any memory location or CPU register to be specified for a bit operation through one of three previous addressing modes (register, register indirect and indexed) while three bits in the OP code specify which of the eight bits is to be manipulated.

ADDRESSING MODE COMBINATIONS

Many instructions include more than one operand (such as arithmetic instructions or loads). In these cases, two types of addressing may be employed. For example, load can use immediate addressing to specify the source and register indirect or indexed addressing to specify the source and register indirect or indexed addressing to specify the destination.

5.3 INSTRUCTION OP CODES

This section describes each of the Z80 instructions and provides tables listing the OP codes for every instruction. In each of these tables the shaded OP codes are identical to those offered in the 8080A CPU. Also shown is the assembly language mnemonic that is used for each instruction. All instruction OP codes are listed in hexadecimal notation. Single byte OP codes require two hex characters while double byte OP codes require four hex characters. The conversion from hex to binary is repeated here for convenience.

Hex	Binary	Decimal	Hex	Binary	Decimal
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	A	1010	10
3	0011	3	B	1011	11
4	0100	4	C	1100	12
5	0101	5	D	1101	13
6	0110	6	E	1110	14
7	0111	7	F	1111	15

Z80 instruction mnemonics consist of an OP code and zero, one or two operands. Instructions in which the operand is implied have no operand. Instructions which have only one logical operand or those in which one operand is invariant (such as the Logical OR instruction) are represented by a one operand mnemonic. Instructions which may have two varying operands are represented by two operand mnemonics.

LOAD AND EXCHANGE

Table 5.3-1 defines the OP code for all of the 8-bit load instructions implemented in the Z80-CPU. Also shown in this table is the type of addressing used for each instruction. The source of the data is found on the top horizontal row while the destination is specified by the left hand column. For example, load register C from register B uses the OP code 48H. In all of the tables the OP code is specified in hexadecimal notation and the 48H (=0100 1000 binary) code is fetched by the CPU from the external memory during M1 time, decoded and then the register transfer is automatically performed by the CPU.

The assembly language mnemonic for this entire group is LD, followed by the destination followed by the source (LD DEST., SOURCE). Note that several combinations of addressing modes are possible. For example, the source may use register addressing and the destination may be register indirect, such as load the memory location pointed to by register HL with the contents of register D. The OP code for this operation would be 72. The mnemonic for this load instruction would be as follows: LD (HL), D

The parentheses around the HL means that the contents of HL are used as a pointer to a memory location. In all Z80 load instruction mnemonics the destination is always listed first, with the source following. The Z80 assembly language has been defined for ease of programming. Every instruction is self documenting and programs written in Z80 language are easy to maintain.

Note in Table 5.3-1 that some load OP codes that are available in the Z80 use two bytes. This is an efficient method of memory utilization since 8, 16, 24 or 32 bit instructions are implemented in the Z80. Thus often utilized instructions such as arithmetic or logical operations are only 8-bits which results in better memory utilization than is achieved with fixed instruction sizes such as 16-bits.

All load instructions using indexed addressing for either the source or destination location actually use three bytes of memory with the third byte being the displacement d. For example a load register E with the operand pointed to by IX with an offset of +8 would be written: LD E, (IX + 8)

The instruction sequence for this in memory would be:

Address A	DD	OP Code
A+1	5F	
A+2	08	Displacement operand

The two extended addressing instructions are also three byte instructions. For example the instruction to load the accumulator with the operand in memory location 6F32H would be written:

LD A, (6F 32H)

and its instruction sequence would be:

Address A	3A	OP Code
A+1	32	low order address
A+2	6F	high order address

Notice that the low order portion of the address is always the first operand.

The load immediate instructions for the general purpose 8-bit registers are two-byte instructions. The instruction load register H with the value 36H would be written:

LD H, 36H

and its sequence would be:

Address A	26	OP Code
A+1	36	Operand

Loading a memory location using indexed addressing for the destination and immediate addressing for the source requires four bytes. For example:

LD (IX - 15), 21H

would appear as:

Address A	DD	OP Code
A+1	36	
A+2	F1	displacement (-15 in signed two's complement)
A+3	21	operand to load

Notice that with any indexed addressing the displacement always follows directly after the OP code.

Table 5.3-2 specifies the 16-bit load operations. This table is very similar to the previous one. Notice that the extended addressing capability covers all register pairs. Also notice that register indirect operations specifying the stack pointer are the PUSH and POP instructions. The mnemonic for these instructions is "PUSH" and "POP". These differ from other 16-bit loads in that the stack pointer is automatically decremented and incremented as each byte is pushed onto or popped from the stack respectively. For example the instruction:

PUSH AF

is a single byte instruction with the OP code of F5H. When this instruction is executed the following sequence is generated:

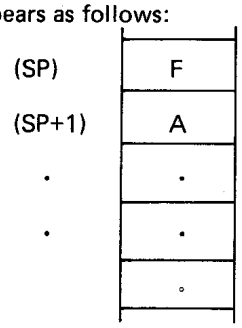
Decrement SP

LD (SP), A

Decrement SP

LD (SP), F

Thus the external stack now appears as follows:



8 BIT LOAD GROUP

DESTINATION			SOURCE																	
			IMPLIED		REGISTER							REG INDIRECT			INDEXED		EXT. ADDR.	IMME.
			I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX + d)	(IY + d)	(nn)	n
REGISTER	A		ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
	B		ED 47	ED 4F	47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
	C		ED 4F	ED 4F	4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
	D		ED 57	ED 57	57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
	E		ED 5F	ED 5F	5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
	H		ED 67	ED 67	67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
	L		ED 6F	ED 6F	6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
REG INDIRECT	(HL)		ED 77	ED 77	77	70	71	72	73	74	75							36 n
	(BC)		ED 02	ED 02	02													
	(DE)		ED 12	ED 12	12													
INDEXED	(IX+d)				DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
	(IY+d)				FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
EXT. ADDR	(nn)				32 n n													
IMPLIED	I				ED 47													
	R				ED 4F													

TABLE 5.3-1

The POP instruction is the exact reverse of a PUSH. Notice that all PUSH and POP instructions utilize a 16-bit operand and the high order byte is always pushed first and popped last. That is a:

PUSH BC is PUSH B then C
PUSH DE is PUSH D then E
PUSH HL is PUSH H then L
POP HL is POP L then H

The instruction using extended immediate addressing for the source obviously requires 2 bytes of data following the OP code. For example:

LD DE, 0659H

will be:

Address A	11	OP Code
A+1	59	Low order operand to register E
A+2	06	High order operand to register D

In all extended immediate or extended addressing modes, the low order byte always appears first after the OP code.

Table 5.3-3 lists the 16-bit exchange instructions implemented in the Z80. OP code 08H allows the programmer to switch between the two pairs of accumulator flag registers while D9H allows the programmer to switch between the duplicate set of six general purpose registers. These OP codes are only one byte in length to absolutely minimize the time necessary to perform the exchange so that the duplicate banks can be used to effect very fast interrupt response times.

BLOCK TRANSFER AND SEARCH

Table 5.3-4 lists the extremely powerful block transfer instructions. All of these i operate with three registers.

HL points to the source location.
DE points to the destination location.
BC is a byte counter.

After the programmer has initialized these three registers, any of these four instru be used. The LDI (Load and Increment) instruction moves one byte from th pointed to by HL to the location pointed to by DE. Register pairs HL and D automatically incremented and are ready to point to the following locations. counter (register pair BC) is also decremented at this time. This instruction is valu blocks of data must be moved but other types of processing are required bet move. The LDIR (Load, increment and repeat) instruction is an extension o instruction. The same load and increment operation is repeated until the by reaches the count of zero. Thus, this single instruction can move any block of dat location to any other.

Note that since 16-bit registers are used, the size of the block can be up to (1K = 1024) long and it can be moved from any location in memory to any othe Furthermore the blocks can be overlapping since there are absolutely no constrai data that is used in the three register pair.

The LDD and LDDR instructions are very similar to the LDI and LDIR. The only is that register pairs HL and DE are decremented after every move so that a bloc starts from the highest address of the designated block rather than the lowest.

16 BIT LOAD GROUP 'LD' 'PUSH' AND 'POP'

SOURCE

DESTINATION REGISTER		REGISTER							IMM. EXT.	EXT. ADDR.	REG. INDIR.
		AF	BC	DE	HL	SP	IX	IY	nn	(nn)	(SP)
REGISTER	AF		C5 F1	D5 F1	E5 F1		DD E5 E1	FD E5 E1	ES 21 nn ES F1 E1	ES 2A nn ES F1 E1	F1
	BC	F5 C1		42 4B	44 4D		DD E5	FD E5	01 n n	ED 4B n n	C1
	DE	F5 D1	50 59		54 5D		DD E5	FD E5	11 n n	ED 5B n n	D1
	HL	F5 E1	60 69	62 6B			DD E5	FD E5	21 n n	2A n n	E1
	SP				F9		DD F9	FD F9	31 n n	ED 7B n n	
	IX	F5 DD E1	C5 DD E1	D5 DD E1	E5 DD E1			FD E5 DD E1	DD 21 n n	DD 2A n n	DD E1
	IY	F5 FD E1	C5 FD E1	D5 FD E1	E5 FD E1		DD E5 FD E1		FD 21 n n	FD 2A n n	FD E1
EXT. ADDR.	(nn)	ES F5 E1 22 (nn) E1	ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n	ES 21 nn 22(nn) E1	ES 2A nn 22(nn) E1	E3 22(nn) E1
REG. IND.	(SP)	F5	C5	D5	E5		DD E5	FD E5	ES 21 nn E3	ES 2A nn E3	

PUSH INSTRUCTIONS →

NOTE: The Push & Pop Instructions adjust the SP after every execution

POP INSTRUCTIONS ↑

TABLE 5.3-2

EXCHANGES 'EX' AND 'EXX'

IMPLIED		IMPLIED ADDRESSING				
		AF'	BC', DE' & HL'	HL	IX	IY
IMPLIED	AF	08		F5 E3 F1	F5 DD E3 F1	F5 FD E3 F1
	BC, DE & HL		D9			
	DE			EB		
REG. INDIR.	(SP)			E3	DD E3	FD E3

TABLE 5.3-3

BLOCK TRANSFER GROUP

			SOURCE: REG. INDIR. (HL)	
DESTINATION	REG. INDIR.	(DE)	ED A0	'LDI' – Load (DE)←(HL) Inc HL & DE, Dec BC
			ED B0	'LDIR,' – Load (DE)←(HL) Inc HL & DE, Dec BC, Repeat until BC = 0
			ED A8	'LDD' – Load (DE)←(HL) Dec HL & DE, Dec BC
			ED B8	'LDDR' – Load (DE)←(HL) Dec HL & DE, Dec BC, Repeat until BC = 0

Reg HL points to source
Reg DE points to destination
Reg BC is byte counter

Table 5.3-4

Table 5.3-5 specifies the OP codes for the four block search instructions. The first, CPI (compare and increment) compares the data in the accumulator, with the contents of the memory location pointed to by register HL. The result of the compare is stored in one of the flag bits (see section 6.0 for a detailed explanation of the flag operations) and the HL register pair is then incremented and the byte counter (register pair BC) is decremented.

The instruction CPIR is merely an extension of the CPI instruction in which the compare is repeated until either a match is found or the byte counter (register pair BC) becomes zero. Thus, this single instruction can search the entire memory for any 8-bit character.

The CPD (Compare and Decrement) and CPDR (Compare, Decrement and Repeat) are similar instructions, their only difference being that they decrement HL after every compare so that they search the memory in the opposite direction. (The search is started at the highest location in the memory block).

It should be emphasized again that these block transfer and compare instructions are extremely powerful in string manipulation applications.

ARITHMETIC AND LOGICAL

Table 5.3-6 lists all of the 8-bit arithmetic operations that can be performed with the accumulator, also listed are the increment (INC) and decrement (DEC) instructions. In all of these instructions, except INC and DEC, the specified 8-bit operation is performed between the data in the accumulator and the source data specified in the table. The result of the operation is placed in the accumulator with the exception of compare (CP) that leaves the accumulator unaffected. All of these operations affect the flag register as a result of the specified operation. (Section 6.0 provides all of the details on how the flags are affected by any instruction type). INC and DEC instructions specify a register or a memory location as both source and destination of the result. When the source operand is addressed using the index registers the displacement must follow directly. With immediate addressing the actual operand will follow directly. for example the instruction:

AND 07H

would appear as:

Address A	E6	OP Code
A+1	07	Operand

BLOCK SEARCH GROUP

SEARCH LOCATION REG. INDIR. (HL)	
ED A1	'CPI' Inc HL, Dec BC
ED B1	'CPIR', Inc HL, Dec BC repeat until BC = 0 or find match
ED A9	'CPD' Dec HL & BC
ED B9	'CPDR' Dec HL & BC Repeat until BC = 0 or find match

HL points to location in memory to be compared with accumulator contents
BC is byte counter

TABLE 5.3-5

Assuming that the accumulator contained the value F3H the result of 03H would be placed in the accumulator:

Acc before operation	1111 0011 = F3H
Operand	0000 0111 = 07H
Result to Acc	0000 0011 = 03H

The Add instruction (ADD) performs a binary add between the data in the source location and the data in the accumulator. The subtract (SUB) does a binary subtraction. When the add with carry is specified (ADC) or the subtract with carry (SBC), then the carry flag is also added or subtracted respectively. The flags and decimal adjust instruction (DAA) in the Z80 (fully described in section 6.0) allow arithmetic operations for:

multiprecision packed BCD numbers

multiprecision signed or unsigned binary numbers

multiprecision two's complement signed numbers

Other instructions in this group are logical and (AND), logical or (OR), exclusive or (XOR) and compare (CP).

There are five general purpose arithmetic instructions that operate on the accumulator or carry flag. These five are listed in Table 5.3-7. The decimal adjust instruction can adjust for subtraction as well as addition, thus making BCD arithmetic operations simple. Note that to allow for this operation the flag N is used. This flag is set if the last arithmetic operation was a subtract. The negate accumulator (NEG) instruction forms the two's complement of the number in the accumulator. Finally notice that a reset carry instruction is not included in the Z80 since this operation can be easily achieved through other instructions such as a logical AND of the accumulator with itself.

Table 5.3-8 lists all of the 16-bit arithmetic operations between 16-bit registers. There are five groups of instructions including add with carry and subtract with carry. ADC and SBC affect all of the flags. These two groups simplify address calculation operations or other 16-bit arithmetic operations.

8 BIT ARITHMETIC AND LOGIC

SOURCE

	REGISTER ADDRESSING							REG. INDIR.	INDEXED		IMMED.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADD w CARRY 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUBTRACT 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SUB w CARRY 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
COMPARE 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INCREMENT 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENT 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

(nn)
ES 21 nn
86 E1
ES 21 nn
8E E1
ES 21 nn
96 E1
ES 21 nn
9E E1
ES 21 nn
A6 E1
ES 21 nn
AE E1
ES 21 nn
B6 E1
ES 21 nn
BE E1
ES 21 nn
34 E1
ES 21 nn
35 E1

TABLE 5.3-6

GENERAL PURPOSE AF OPERATIONS

Decimal Adjust Acc, 'DAA'	27
Complement Acc, 'CPL'	2F
Negate Acc, 'NEG' (2's complement)	ED 44
Complement Carry Flag, 'CCF'	3F
Set Carry Flag, 'SCF'	37

TABLE 5.3-7

16 BIT ARITHMETIC

DESTINATION			SOURCE BC	DE	HL	SP	IX	IY
	'ADD'	HL	09	19	29	39	DD 29	FD 29
		IX	DD 09	DD 19	DD 29	DD 39	DD 29	DD 29
		IY	FD 09	FD 19	FD 29	FD 39	FD 29	FD 29
	ADD WITH CARRY AND SET FLAGS 'ADC'	HL	ED 4A	ED 5A	ED 6A	ED 7A	ED 7A	ED 7A
	SUB WITH CARRY AND SET FLAGS 'SBC'	HL	ED 42	ED 52	ED 62	ED 72	ED 72	ED 72
	INCREMENT 'INC.'		03	13	23	33	DD 23	FD 23
	DECREMENT 'DEC'		0B	1B	2B	3B	DD 2B	FD 2B

TABLE 5.3-8

ROTATE AND SHIFT

A major capability of the Z80 is its ability to rotate or shift data in the accumulator, any general purpose register, or any memory location. All of the rotate and shift OP codes are shown in Table 5.3-9. Also included in the Z80 are arithmetic and logical shift operations. These operations are useful in an extremely wide range of applications including integer multiplication and division. Two BCD digit rotate instructions (RRD and RLD) allow a digit in the accumulator to be rotated with the two digits in a memory location pointed to by register pair HL. (See Figure 5.3-9). These instructions allow for efficient BCD arithmetic.

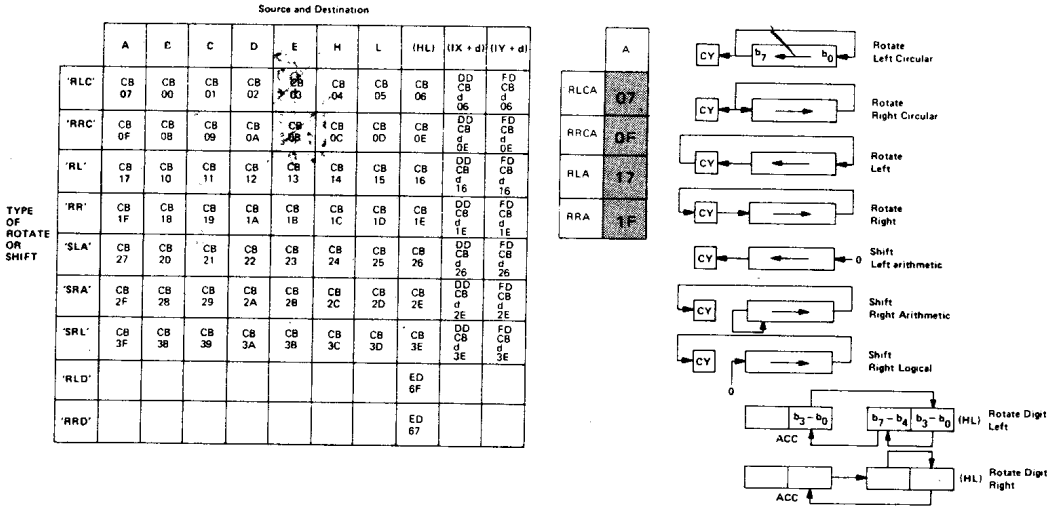
BIT MANIPULATION

The ability to set, reset and test individual bits in a register or memory location is needed in almost every program. These bits may be flags in a general purpose software routine, indications of external control conditions or data packed into memory locations to make memory utilization more efficient.

The Z80 has the ability to set, reset or test any bit in the accumulator, any general purpose register or any memory location with a single instruction. Table 5.3-10 lists the 240 instructions that are available for this purpose. Register addressing can specify the accumulator or any general purpose register on which the operation is to be performed. Register indirect and indexed addressing are available to operate on external memory locations. Bit test operations set the zero flag (Z) if the tested bit is a zero. (Refer to section 6.0 for further explanation of flag operation).

JUMP, CALL AND RETURN

Figure 5.3-11 lists all of the jump, call and return instructions implemented in the Z80 CPU. A jump is a branch in a program where the program counter is loaded with the 16-bit value as specified by one of the three available addressing modes (Immediate Extended, Relative or Register Indirect). Notice that the jump group has several different conditions that can be specified to be met before the jump will be made. If these conditions are not met, the program merely continues with the next sequential instruction. The conditions are all dependent on the data in the flag register. (Refer to section 6.0 for details on the flag register). The immediate extended addressing is used to jump to any location in the memory. This instruction requires three bytes (two to specify the 16-bit address) with the low order address byte first followed by the high order address byte.



Source and Destination

TYPE OF ROTATE OR SHIFT	A	B	C	D	E	H	L	(HL)	(IX + d)	(IY + d)
'RLC'	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
'RRC'	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
'RL'	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
'RR'	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
'SLA'	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
'SRA'	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
'SRL'	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
'RLD'								ED 6F		
'RRD'								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F

Rotate Left Circular
Rotate Right Circular
Rotate Left
Rotate Right
Shift Left arithmetic
Shift Right Arithmetic
Shift Right Logical
Rotate Digit Left
Rotate Digit Right

TABLE 5.3-9

For example an unconditional Jump to memory location 3E32H would be:

Address A	C3	OP Code
A+1	32	Low order address
A+2	3E	High order address

The relative jump instruction uses only two bytes, the second byte is a signed two's complement displacement from the existing PC. This displacement can be in the range of +129 to -126 and is measured from the address of the instruction OP code.

Three types of register indirect jumps are also included. These instructions are implemented by loading the register pair HL or one of the index registers IX or IY directly into the PC. This capability allows for program jumps to be a function of previous calculations.

A call is a special form of a jump where the address of the byte following the call instruction is pushed onto the stack before the jump is made. A return instruction is the reverse of a call because the data on the top of the stack is popped directly into the PC to form a jump address. The call and return instructions allow for simple subroutine and interrupt handling. Two special return instructions have been included in the Z80 family of components. The return from interrupt instruction (RETI) and the return from non-maskable interrupt (RETN) are treated in the CPU as an unconditional return identical to the OP code C9H. The difference is that (RETI) can be used at the end of an interrupt routine and all Z80 peripheral chips will recognize the execution of this instruction for proper control of nested priority interrupt handling. This instruction coupled with the Z80 peripheral devices implementation simplifies the normal return from nested interrupt. Without this feature the following software sequence would be necessary to inform the interrupting device that the interrupt routine is completed:

BIT MANIPULATION GROUP

	BIT	REGISTER ADDRESSING							REG. INDIR.	INDEXED	
		A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
TEST 'BIT'	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	GB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RESET BIT 'RES'	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET BIT 'SET'	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

TABLE 5.3-10

Disable Interrupt	— prevent interrupt before routine is exited.
LD A, n OUT n, A	— notify peripheral that service routine is complete
Enable Interrupt	
Return	

This seven byte sequence can be replaced with the three byte EI RETI instruction sequence in the Z80. This is important since interrupt service time often must be minimized.

To facilitate program loop control the instruction DJNZ e can be used advantageously. This two byte, relative jump instruction decrements the B register and the jump occurs if the B register has not been decremented to zero. The relative displacement is expressed as a signed two's complement number. A simple example of its use might be:

Address	Instruction	Comments
N, N+1	LD B, 7	; set B register to count of 7
N + 2 to N + 9	(Perform a sequence of instructions)	; loop to be performed 7 times
N + 10, N + 11	DJNZ -10	; to jump from N + 12 to N + 2
N + 12	(Next Instruction)	

JUMP, CALL AND RETURN GROUP

			CONDITION									
			UN-COND.	CARRY	NON CARRY	ZERO	NON ZERO	PARITY EVEN	PARITY ODD	SIGN NEG	SIGN POS	REG B≠0
JUMP 'JP'	IMMED. EXT.	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JUMP 'JR'	RELATIVE	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JUMP 'JP'	REG. INDIR.	(HL)*	E9									
JUMP 'JP'		(IX)	DD E9									
JUMP 'JP'		(IY)	FD E9									
'CALL'	IMMED. EXT.	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DECREMENT B, JUMP IF NON ZERO 'DJNZ'	RELATIVE	PC+e										10 e-2
RETURN 'RET'	REGISTER INDIR.	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETURN FROM INT 'RETI'	REG. INDIR.	(SP) (SP+1)	ED 4D									
RETURN FROM NON MASKABLE INT 'RETN'	REG. INDIR.	(SP) (SP+1)	ED 45									

NOTE–CERTAIN FLAGS HAVE MORE THAN ONE PURPOSE. REFER TO SECTION 6.0 FOR DETAILS

TABLE 5.3-11

Table 5.3-12 lists the eight OP codes for the restart instruction. This instruction is a single byte call to any of the eight addresses listed. The simple mnemonic for these eight calls is also shown. The value of this instruction is that frequently used routines can be called with this instruction to minimize memory usage.

RESTART GROUP

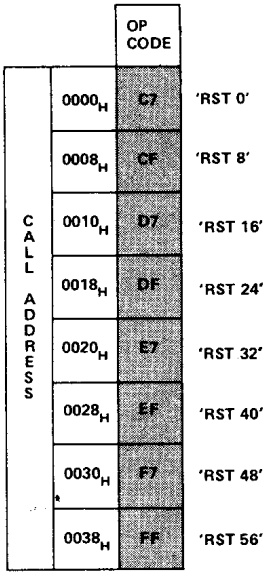


TABLE 5.3-12

INPUT/OUTPUT

The Z80 has an extensive set of Input and Output instructions as shown in table 5.3-13 and table 5.3-14. The addressing of the input or output device can be either absolute or register indirect, using the C register. Notice that in the register indirect addressing mode data can be transferred between the I/O devices and any of the internal registers. In addition eight block transfer instructions have been implemented. These instructions are similar to the memory block transfers except that they use register pair HL for a pointer to the memory source (output commands) or destination (input commands) while register B is used as a byte counter. Register C holds the address of the port for which the input or output command is desired. Since register B is eight bits in length, the I/O block transfer command handles up to 256 bytes.

In the instructions IN A, n and OUT n, A an I/O device address n appears in the lower half of the address bus (A_0-A_7) while the accumulator content is transferred in the upper half of the address bus. In all register indirect input output instructions, including block I/O transfers the content of register C is transferred to the lower half of the address bus (device address) while the content of register B is transferred to the upper half of the address bus.

INPUT GROUP

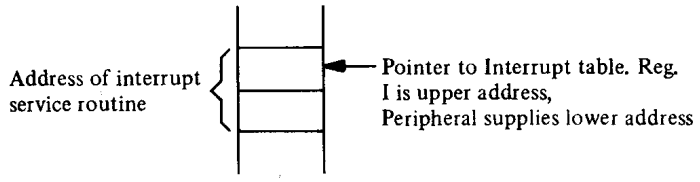
INPUT DESTINATION				PORT ADDRESS: IMMED. (n)	PORT ADDRESS: REG. INDIR. (C)
INPUT 'IN'	REG ADDRESSING	A		DB n	ED 78
		B		F5 78 DB n F1	ED 40
		C		F5 79 DB n F1	ED 48
		D		F5 7A DB n F1	ED 50
		E		F5 7B DB n F1	ED 58
		H		F5 7C DB n F1	ED 60
		L		F5 7D DB n F1	ED 68
'INI' – INPUT & Inc HL, Dec B	REG, INDIR	(HL)			ED A2
'INIR'– INP, Inc HL, Dec B, REPEAT IF B≠0					ED B2
'IND'– INPUT & Dec HL, Dec B					ED AA
'INDR'– INPUT, Dec HL, Dec B, REPEAT IF B≠0					ED BA

BLOCK INPUT COMMANDS

TABLE 5.3-13

CPU CONTROL GROUP

The final table, table 5.3-15 illustrates the six general purpose CPU control instructions. The NOP is a do-nothing instruction. The HALT instruction suspends CPU operation until a subsequent interrupt is received, while the DI and EI are used to lock out and enable interrupts. The three interrupt mode commands set the CPU into any of the three available interrupt response modes as follows. If mode zero is set the interrupting device can insert any instruction on the data bus and allow the CPU to execute it. Mode 1 is a simplified mode where the CPU automatically executes a restart (RST) to location 0038H so that no external hardware is required. (The old PC content is pushed onto the stack). Mode 2 is the most powerful in that it allows for an indirect call to any location in memory. With this mode the CPU forms a 16-bit memory address where the upper 8-bits are the content of register I and the lower 8-bits are supplied by the interrupting device. This address points to the first of two sequential bytes in a table where the address of the service routine is located. The CPU automatically obtains the starting address and performs a CALL to this address.


Address of interrupt service routine
Pointer to Interrupt table. Reg. I is upper address, Peripheral supplies lower address

OUTPUT GROUP

			SOURCE							
			REGISTER							REG. IND.
			A	B	C	D	E	H	L	(HL)
'OUT'	IMMED.	n	D3 n	F5 78 D3 n F1	F5 79 D3 n F1	F5 7A D3 n F1	F5 7B D3 n F1	F5 7C D3 n F1	F5 7D D3 n F1	F5 7E D3 n F1
	REG. IND.	(C)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69	F5 7E ED 79 F1
'OUTI' – OUTPUT Inc HL, Dec b	REG. IND.	(C)								ED A3
'OTIR' – OUTPUT, Inc HL, Dec B, REPEAT IF B≠0	REG. IND.	(C)								ED B3
'OUTD' – OUTPUT Dec HL & B	REG. IND.	(C)								ED AB
'OTDR' – OUTPUT, Dec HL & B, REPEAT IF B≠0	REG. IND.	(C)								ED BB

BLOCK OUTPUT COMMANDS

PORT DESTINATION ADDRESS

TABLE 5.3-14

MISCELLANEOUS CPU CONTROL

'NOP'	00	
'HALT'	76	
DISABLE INT '(DI)'	F3	
ENABLE INT '(EI)'	FB	
SET INT MODE 0 'IM0'	ED 46	8080A MODE
SET INT MODE 1 'IM1'	ED 56	CALL TO LOCATION 0038_H
SET INT MODE 2 'IM2'	ED 5E	INDIRECT CALL USING REGISTER I AND 8 BITS FROM INTERRUPTING DEVICE AS A POINTER.

TABLE 5.3-15

(Page intentionally left blank)

6.0 FLAGS

Each of the two Z80-CPU Flag registers contains six bits of information which are set or reset by various CPU operations. Four of these bits are testable; that is, they are used as conditions for jump, call or return instructions. For example a jump may be desired only if a specific bit in the flag register is set. The four testable flag bits are:

1) Carry Flag (C) — This flag is the carry from the highest order bit of the accumulator. For example, the carry flag will be set during an add instruction where a carry from the highest bit of the accumulator is generated. This flag is also set if a borrow is generated during a subtraction instruction. The shift and rotate instructions also affect this bit.

2) Zero Flag (Z) — This flag is set if the result of the operation loaded a zero into the accumulator. Otherwise it is reset.

3) Sign Flag(S) — This flag is intended to be used with signed numbers and it is set if the result of the operation was negative. Since bit 7 (MSB) represents the sign of the number (A negative number has a 1 in bit 7), this flag stores the state of bit 7 in the accumulator.

4) Parity/Overflow Flag(P/V) — This dual purpose flag indicates the parity of the result in the accumulator when logical operations are performed (such as AND A, B) and it represents overflow when signed two's complement arithmetic operations are performed. The Z80 overflow flag indicates that the two's complement number in the accumulator is in error since it has exceeded the maximum possible (+127) or is less than the minimum possible (—128) number that can be represented two's complement notation. For example consider adding:

```
+120 =   0111 1000
+105 =   0110 1001
  C = 0  1110 0001 = -95 (wrong) Overflow has occurred;
```

Here the result is incorrect. Overflow has occurred and yet there is no carry to indicate an error. For this case the overflow flag would be set. Also consider the addition of two negative numbers:

```
-5  =    1111 1011
-16 =    1111 0000
  C = 1  1110 1011 = -21 correct
```

Notice that the answer is correct but the carry is set so that this flag can not be used as an overflow indicator. In this case the overflow would not be set.

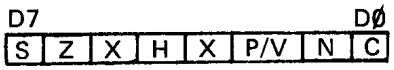
For logical operations (AND, OR, XOR) this flag is set if the parity of the result is even and it is reset if it is odd.

There are also two non-testable bits in the flag register. Both of these are used for BCD arithmetic. They are:

1) Half carry(H) — This is the BCD carry or borrow result from the least significant four bits of operation. When using the DAA (Decimal Adjust Instruction) this flag is used to correct the result of a previous packed decimal add or subtract.

2) Add/Subtract Flag (N) — Since the agorithim for correcting BCD operations is different for addition or subtraction, this flag is used to specify what type of instruction was executed last so that the DAA operation will be correct for either addition or subtraction.

The Flag register can be accessed by the programmer and its format is as follows:



X means flag is indeterminate.

Table 6.0-1 lists how each flag bit is affected by various CPU instructions. In this table a '•' indicates that the instruction does not change the flag, an 'X' means that the flag goes to an indeterminate state, an '0' means that it is reset, a '1' means that it is set and the symbol ↕ indicates that it is set or reset according to the previous discussion. Note that any instruction not appearing in this table does not affect any of the flags.

Table 6.0-1 includes a few special cases that must be described for clarity. Notice that the block search instruction sets the Z flag if the last compare operation indicated a match between the source and the accumulator data. Also, the parity flag is set if the byte counter (register pair BC) is not equal to zero. This same use of the parity flag is made with the block move instructions. Another special case is during block input or output instructions, here the Z flag is used to indicate the state of register B which is used as a byte counter. Notice that when the I/O block transfer is complete, the zero flag will be reset to a zero (i.e. B=0) while in the case of a block move command the parity flag is reset when the operation is complete. A final case is when the refresh or I register is loaded into the accumulator, the interrupt enable flip flop is loaded into the parity flag so that the complete state of the CPU can be saved at any time.

SUMMARY OF FLAG OPERATION

Instruction	D7 S	Z		H		P/V	N	D0 C	Comments
ADD A,s; ADC A,s	↕	↕	X	↕	X	V	0	↕	8-bit add or add with carry
SUB,s; SBCA,s; CP,s; NEG	↕	↕	X	↕	X	V	1	↕	8-bit subtract, subtract with carry, compare and negate accumulator
AND s	↕	↕	X	1	X	P	0	0	} Logical operations
OR s; XOR s	↕	↕	X	0	X	P	0	0	} Logical operations
INC s	↕	↕	X	↕	X	V	0	•	8-bit increment
DEC s	↕	↕	X	↕	X	V	1	•	8-bit decrement
ADD DD, SS	•	•	X	X	X	•	0	↕	16-bit add
ADC HL, SS	↕	↕	X	X	X	V	0	↕	16-bit add with carry
SBC HL, SS	↕	↕	X	X	X	V	1	↕	16-bit subtract with carry
RLA; RLCA; RRA; RRCA	•	•	X	0	X	•	0	↕	Rotate accumulator
RL s; RLC s; RR s; RRC s; SLA s; SRA s; SRL s	↕	↕	X	0	X	P	0	↕	Rotate and shift locations
RLD; RRD	↕	↕	X	0	X	P	0	•	Rotate digit left and right
DAA	↕	↕	X	↕	X	P	•	↕	Decimal adjust accumulator
CPL	•	•	X	1	X	•	1	•	Complement accumulator
SCF	•	•	X	0	X	•	0	1	Set carry
CCF	•	•	X	X	X	•	0	↕	Complement carry
IN r, (C)	↕	↕	X	0	X	P	0	•	Input register indirect
INI; INO; OUTI; OUTD	X	↕	X	X	X	X	1	X	} Block input and output
INIR; INDR; OTIR; OTDR	X	1	X	X	X	X	1	X	} Z = 0 if B ≠ 0 otherwise Z = 1
LDI; LDD	X	X	X	0	X	↕	0	•	} Block transfer instructions
LDIR; LDDR	X	X	X	0	X	0	0	•	} P/V = 1 if BC ≠ 0, otherwise P/V = 0
CPI; CPIR; CPD; CPDR	↕	↕	X	↕	X	↕	1	•	Block search instructions Z = 1 if A = (HL), otherwise Z = 0 P/V = 1 if BC ≠ 0, otherwise P/V = 0
LD A, I; LD A, R	↕	↕	X	0	X	IFF	0	•	The content of the interrupt enable flip-flop (IFF) is copied into the P/V flag
BIT b, s	X	↕	X	1	X	X	0	•	The state of bit b of location s is copied into the Z flag

The following notation is used in this table:

SYMBOL	OPERATION
C	Carry/link flag. C=1 if the operation produced a carry from the MSB of the operand or result.
Z	Zero flag. Z=1 if the result of the operation is zero.
S	Sign flag. S=1 if the MSB of the result is one.
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V=1 if the result of the operation is even, P/V=0 if result is odd. If P/V holds overflow, P/V=1 if the result of the operation produced an overflow.
H	Half-carry flag. H=1 if the add or subtract operation produced a carry into or borrow from bit 4 of the accumulator.
N	Add/Subtract flag. N=1 if the previous operation was a subtract.
	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtraction using operands with packed BCD format.
	The flag is affected according to the result of the operation.
•	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation.
X	The flag is a "don't care".
V	P/V flag affected according to the overflow result of the operation.
P	P/V flag affected according to the parity result of the operation.
r	Any one of the CPU registers A, B, C, D, E, H, L.
s	Any 8-bit location for all the addressing modes allowed for the particular instruction.
ss	Any 16-bit location for all the addressing modes allowed for that instruction.
ii	Any one of the two index registers IX or IY.
R	Refresh counter.
n	8-bit value in range <0, 255>
nn	16-bit value in range <0, 65535>

TABLE 6.0-1

(Page intentionally left blank)

7.0 SUMMARY OF OP CODES AND EXECUTION TIMES

The following section gives a summary of the Z80 instruction set. The instructions are logically arranged into groups as shown on Tables 7.0-1 through 7.0-11. Each table shows the assembly language mnemonic OP code, the actual OP code, the symbolic operation, the content of the flag register following the execution of each instruction, the number of bytes required for each instruction as well as the number of memory cycles and the total number of T states (external clock periods) required for the fetching and execution of each instruction. Care has been taken to make each table self-explanatory without requiring any cross reference with the text or other tables.

8-BIT LOAD GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments r, s	Reg.
LD r, s	r ← s	•	•	X	•	X	•	•	•	01 r s		1	1	4	r, s	Reg.
LD r, n	r ← n	•	•	X	•	X	•	•	•	00 r 110		2	2	7	000	B
										← n →					001	C
LD r, (HL)	r ← (HL)	•	•	X	•	X	•	•	•	01 r 110		1	2	7	010	D
LD r, (IX+d)	r ← (IX+d)	•	•	X	•	X	•	•	•	11 011 101	DD	3	5	19	011	E
										01 r 110					100	H
										← d →					101	L
LD r, (IY+d)	r ← (IY+d)	•	•	X	•	X	•	•	•	11 111 101	FD	3	5	19	111	A
										01 r 110						
										← d →						
LD (HL), r	(HL) ← r	•	•	X	•	X	•	•	•	01 110 r		1	2	7		
LD (IX+d), r	(IX+d) ← r	•	•	X	•	X	•	•	•	11 011 101	DD	3	5	19		
										01 110 r						
										← d →						
LD (IY+d), r	(IY+d) ← r	•	•	X	•	X	•	•	•	11 111 101	FD	3	5	19		
										01 110 r						
										← d →						
LD (HL), n	(HL) ← n	•	•	X	•	X	•	•	•	00 110 110	36	2	3	10		
										← n →						
LD (IX+d), n	(IX+d) ← n	•	•	X	•	X	•	•	•	11 011 101	DD	4	5	19		
										00 110 110	36					
										← d →						
										← n →						
LD (IY+d), n	(IY+d) ← n	•	•	X	•	X	•	•	•	11 111 101	FD	4	5	19		
										00 110 110	36					
										← d →						
										← n →						
LD A, (BC)	A ← (BC)	•	•	X	•	X	•	•	•	00 001 010	0A	1	2	7		
LD A, (DE)	A ← (DE)	•	•	X	•	X	•	•	•	00 011 010	1A	1	2	7		
LD A, (nn)	A ← (nn)	•	•	X	•	X	•	•	•	00 111 010	3A	3	4	13		
										← n →						
										← n →						
LD (BC), A	(BC) ← A	•	•	X	•	X	•	•	•	00 000 010	02	1	2	7		
LD (DE), A	(DE) ← A	•	•	X	•	X	•	•	•	00 010 010	12	1	2	7		
LD (nn), A	(nn) ← A	•	•	X	•	X	•	•	•	00 110 010	32	3	4	13		
										← n →						
										← n →						
LD A, I	A ← I	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9		
										01 010 111	57					
LD A, R	A ← R	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9		
										01 011 111	5F					
LD I, A	I ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9		
										01 000 111	47					
LD R, A	R ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9		
										01 001 111	4F					

Notes: r, s means any of the registers A, B, C, D, E, H, L
IFF the content of the interrupt enable flip-flop (IFF) is copied into the P/V flag

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

Table 7.0-1

16-BIT LOAD GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD dd, nn	dd ← nn	•	•	X	•	X	•	•	•	00 dd0 001 ← n → ← n →		3	3	10	dd Pair 00 BC 01 DE 10 HL 11 SP
LD IX, nn	IX ← nn	•	•	X	•	X	•	•	•	11 011 101 00 100 001 ← n → ← n →	DD 21	4	4	14	
LD IY, nn	IY ← nn	•	•	X	•	X	•	•	•	11 111 101 00 100 001 ← n → ← n →	FD 21	4	4	14	
LD HL, (nn)	H ← (nn+1) L ← (nn)	•	•	X	•	X	•	•	•	00 101 010 ← n → ← n →	2A	3	5	16	
LD dd, (nn)	dd_H ← (nn+1) dd_L ← (nn)	•	•	X	•	X	•	•	•	11 101 101 01 dd1 011 ← n → ← n →	ED	4	6	20	
LD IX, (nn)	IX_H ← (nn+1) IX_L ← (nn)	•	•	X	•	X	•	•	•	11 011 101 00 101 010 ← n → ← n →	DD 2A	4	6	20	
LD IY, (nn)	IY_H ← (nn+1) IY_L ← (nn)	•	•	X	•	X	•	•	•	11 111 101 00 101 010 ← n → ← n →	FD 2A	4	6	20	
LD (nn), HL	(nn+1) ← H (nn) ← L	•	•	X	•	X	•	•	•	00 100 010 ← n → ← n →	22	3	5	16	
LD (nn), dd	(nn+1) ← dd_H (nn) ← dd_L	•	•	X	•	X	•	•	•	11 101 101 01 dd0 011 ← n → ← n →	ED	4	6	20	
LD (nn), IX	(nn+1) ← IX_H (nn) ← IX_L	•	•	X	•	X	•	•	•	11 011 101 00 100 010 ← n → ← n →	DD 22	4	6	20	
LD (nn), IY	(nn+1) ← IY_H (nn) ← IY_L	•	•	X	•	X	•	•	•	11 111 101 00 100 010 ← n → ← n →	FD 22	4	6	20	
LD SP, HL	SP ← HL	•	•	X	•	X	•	•	•	11 111 001	F9	1	1	6	
LD SP, IX	SP ← IX	•	•	X	•	X	•	•	•	11 011 101 11 111 001	DD F9	2	2	10	
LD SP, IY	SP ← IY	•	•	X	•	X	•	•	•	11 111 101 11 111 001	FD F9	2	2	10	
PUSH qq	(SP-2) ← qq_L (SP-1) ← qq_H	•	•	X	•	X	•	•	•	11 qq0 101		1	3	11	qq Pair 00 BC 01 DE 10 HL 11 AF
PUSH IX	(SP-2) ← IX_L (SP-1) ← IX_H	•	•	X	•	X	•	•	•	11 011 101 11 100 101	DD E5	2	4	15	
PUSH IY	(SP-2) ← IY_L (SP-1) ← IY_H	•	•	X	•	X	•	•	•	11 111 101 11 100 101	FD E5	2	4	15	
POP qq	qq_H ← (SP+1) qq_L ← (SP)	•	•	X	•	X	•	•	•	11 qq0 001		1	3	10	
POP IX	IX_H ← (SP+1) IX_L ← (SP)	•	•	X	•	X	•	•	•	11 011 101 11 100 001	DD E1	2	4	14	
POP IY	IY_H ← (SP+1) IY_L ← (SP)	•	•	X	•	X	•	•	•	11 111 101 11 100 001	FD E1	2	4	14	

Notes: dd is any of the register pairs BC, DE, HL, SP
qq is any of the register pairs AF, BC, DE, HL
$(PAIR)_H$, $(PAIR)_L$ refer to high order and low order eight bits of the register pair respectively.
e.g. BC_L = C, AF_H = A

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ flag is affected according to the result of the operation.

Table 7.0-2

EXCHANGE GROUP AND BLOCK TRANSFER AND SEARCH GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE↔HL	•	•	X	•	X	•	•	•	11 101 011	EB	1	1	4	
EX AF, AF'	AF↔AF'	•	•	X	•	X	•	•	•	00 001 000	08	1	1	4	
EXX	(BC↔BC' DE↔DE' HL↔HL')	•	•	X	•	X	•	•	•	11 011 001	D9	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ↔(SP+1) L ↔(SP)	•	•	X	•	X	•	•	•	11 100 011	E3	1	5	19	
EX (SP), IX	IX_H ↔(SP+1) IX_L ↔(SP)	•	•	X	•	X	•	•	•	11 011 101 11 100 011	DD E3	2	6	23	
EX (SP), IY	IY_H ↔(SP+1) IY_L ↔(SP)	•	•	X	•	X	•	•	•	11 111 101 11 100 011	FD E3	2	6	23	
LDI	(DE)←(HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	X	0	X	① ↕	0	•	11 101 101 10 100 000	ED A0	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE)←(HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 10 110 000	ED B0	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
LDD	(DE)←(HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	X	0	X	① ↕	0	•	11 101 101 10 101 000	ED A8	2	4	16	
LDDR	(DE)←(HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 10 111 000	ED B8	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A – (HL) HL ← HL+1 BC ← BC-1	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 100 001	ED A1	2	4	16	
CPIR	A – (HL) HL ← HL+1 BC ← BC-1 Repeat until A = (HL) or BC = 0	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 110 001	ED B1	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A – (HL) HL ← HL-1 BC ← BC-1	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 101 001	ED A9	2	4	16	
CPDR	A – (HL) HL ← HL-1 BC ← BC-1 Repeat until A = (HL) or BC = 0	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 111 001	ED B9	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)

Notes: ① P/V flag is 0 if the result of BC-1 = 0, otherwise P/V = 1
② Z flag is 1 if A = (HL), otherwise Z = 0.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

Table 7.0-3

8-BIT ARITHMETIC AND LOGICAL GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD A, r	A ← A + r	‡	‡	X	‡	X	V	0	‡	10 [000] r		1	1	4	r Reg.
ADD A, n	A ← A + n	‡	‡	X	‡	X	V	0	‡	11 [000] 110		2	2	7	000 B
										← n →					001 C
															010 D
ADD A, (HL)	A ← A+(HL)	‡	‡	X	‡	X	V	0	‡	10 [000] 110		1	2	7	011 E
ADD A, (IX+d)	A←A+(IX+d)	‡	‡	X	‡	X	V	0	‡	11 011 101	DD	3	5	19	100 H
										10 [000] 110					101 L
										← d →					111 A
ADD A, (IY+d)	A←A+(IY+d)	‡	‡	X	‡	X	V	0	‡	11 111 101	FD	3	5	19	
										10 [000] 110					
										← d →					
ADC A, s	A←A+s+CY	‡	‡	X	‡	X	V	0	‡	[001]					s is any of r, n,
SUB s	A←A - s	‡	‡	X	‡	X	V	1	‡	[010]					(HL), (IX+d),
SBC A, s	A←A - s - CY	‡	‡	X	‡	X	V	1	‡	[011]					(IY+d) as shown for
AND s	A←A ∧ s	‡	‡	X	1	X	P	0	0	[100]					ADD instruction.
OR s	A←A ∨ s	‡	‡	X	0	X	P	0	0	[110]					The indicated bits
XOR s	A←A ⊕ s	‡	‡	X	0	X	P	0	0	[101]					replace the [000] in
CP s	A - s	‡	‡	X	‡	X	V	1	‡	[111]					the ADD set above.
INC r	r ← r + 1	‡	‡	X	‡	X	V	0	•	00 r [100]		1	1	4	
INC (HL)	(HL)←(HL)+1	‡	‡	X	‡	X	V	0	•	00 110 [100]		1	3	11	
INC (IX+d)	(IX+d) ← (IX+d)+1	‡	‡	X	‡	X	V	0	•	11 011 101	DD	3	6	23	
										00 110 [100]					
										← d →					
INC (IY+d)	(IY+d) ← (IY+d)+1	‡	‡	X	‡	X	V	0	•	11 111 101	FD	3	6	23	
										00 110 [100]					
										← d →					
DEC s	s ← s - 1	‡	‡	X	‡	X	V	1	•	[101]					s is any of r, (HL),
															(IX+d), (IY+d) as
															shown for INC.
															DEC same format
															and states as INC.
															Replace [100] with
															[101] in OP Code.

Notes: The V symbol in the P/V flag column indicates that the P/V flag contains the overflow of the result of the operation. Similarly the P symbol indicates parity. V = 1 means overflow, V = 0 means not overflow, P = 1 means parity of the result is even, P = 0 means parity of the result is odd.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
‡ = flag is affected according to the result of the operation.

Table 7.0-4

GENERAL PURPOSE ARITHMETIC AND CPU CONTROL GROUPS

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
DAA	Converts acc, content into packed BCD following add or subtract with packed BCD operands	‡	‡	X	‡	X	P	•	‡	00 100 111	27	1	1	4	Decimal adjust accumulator
CPL	$A \leftarrow \overline{A}$	•	•	X	1	X	•	1	•	00 101 111	2F	1	1	4	Complement accumulator (One's complement)
NEG	$A \leftarrow \overline{A}+1$	‡	‡	X	‡	X	V	1	‡	11 101 101 01 000 100	ED 44	2	2	8	Negate acc, (two's complement)
CCF	$CY \leftarrow \overline{CY}$	•	•	X	X	X	•	0	‡	00 111 111	3F	1	1	4	Complement carry flag
SCF	CY ← 1	•	•	X	0	X	•	0	1	00 110 111	37	1	1	4	Set carry flag
NOP	No operation	•	•	X	•	X	•	•	•	00 000 000	00	1	1	4	
HALT	CPU halted	•	•	X	•	X	•	•	•	01 110 110	76	1	1	4	
DI*	IFF ← 0	•	•	X	•	X	•	•	•	11 110 011	F3	1	1	4	
EI*	IFF ← 1	•	•	X	•	X	•	•	•	11 111 011	FB	1	1	4	
IM 0	Set interrupt mode 0	•	•	X	•	X	•	•	•	11 101 101 01 000 110	ED 46	2	2	8	
IM 1	Set interrupt mode 1	•	•	X	•	X	•	•	•	11 101 101 01 010 110	ED 56	2	2	8	
IM 2	Set interrupt mode 2	•	•	X	•	X	•	•	•	11 101 101 01 011 110	ED 5E	2	2	8	

Notes: IFF indicates the interrupt enable flip-flop
CY indicates the carry flip-flop.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

*Interrupts are not sampled at the end of EI or DI

Table 7.0-5

16-BIT ARITHMETIC GROUP

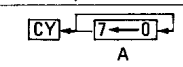
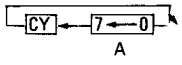
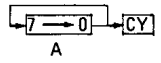
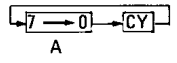
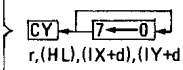
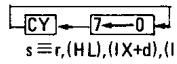
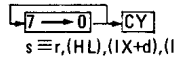
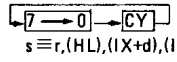
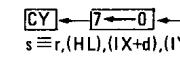
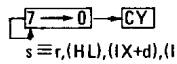
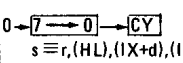
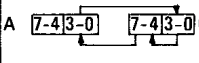
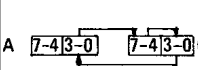
Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD HL, ss	HL ← HL+ss	•	•	X	X	X	•	0	‡	00 ss1 001		1	3	11	ss Reg. 00 BC
ADC HL, ss	HL ← HL+ss+CY	‡	‡	X	X	X	V	0	‡	11 101 101 01 ss1 010	ED	2	4	15	01 DE 10 HL 11 SP
SBC HL, ss	HL ← HL-ss-CY	‡	‡	X	X	X	V	1	‡	11 101 101 01 ss0 010	ED	2	4	15	
ADD IX, pp	IX ← IX + pp	•	•	X	X	X	•	0	‡	11 011 101 00 pp1 001	DD	2	4	15	pp Reg. 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY ← IY + rr	•	•	X	X	X	•	0	‡	11 111 101 00 rr1 001	FD	2	4	15	rr Reg. 00 BC 01 DE 10 IY 11 SP
INC ss	ss ← ss + 1	•	•	X	•	X	•	•	•	00 ss0 011		1	1	6	
INC IX	IX ← IX + 1	•	•	X	•	X	•	•	•	11 011 101 00 100 011	DD 23	2	2	10	
INC IY	IY ← IY + 1	•	•	X	•	X	•	•	•	11 111 101 00 100 011	FD 23	2	2	10	
DEC ss	ss ← ss - 1	•	•	X	•	X	•	•	•	00 ss1 011		1	1	6	
DEC IX	IX ← IX - 1	•	•	X	•	X	•	•	•	11 011 101 00 101 011	DD 2B	2	2	10	
DEC IY	IY ← IY - 1	•	•	X	•	X	•	•	•	11 111 101 00 101 011	FD 2B	2	2	10	

Notes: ss is any of the register pairs BC, DE, HL, SP
pp is any of the register pairs BC, DE, IX, SP
rr is any of the register pairs BC, DE, IY, SP.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
‡ = flag is affected according to the result of the operation.

Table 7.0-6

ROTATE AND SHIFT GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
RLCA		•	•	X	0	X	•	0	‡	00 000 111	07	1	1	4	Rotate left circular accumulator
RLA		•	•	X	0	X	•	0	‡	00 010 111	17	1	1	4	Rotate left accumulator
RRCA		•	•	X	0	X	•	0	‡	00 001 111	0F	1	1	4	Rotate right circular accumulator
RRA		•	•	X	0	X	•	0	‡	00 011 111	1F	1	1	4	Rotate right accumulator
RLC r		‡	‡	X	0	X	P	0	‡	11 001 011 00 000 r	CB	2	2	8	Rotate left circular register r
RLC (HL)		‡	‡	X	0	X	P	0	‡	11 001 011 00 000 110	CB	2	4	15	r Reg. 000 B 001 C
RLC (IX+d)		‡	‡	X	0	X	P	0	‡	11 011 101 11 001 011 ← d → 00 000 110	DD CB	4	6	23	010 D 011 E 100 H 101 L 111 A
RLC (IY+d)	r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	11 111 101 11 001 011 ← d → 00 000 110	FD CB	4	6	23	
RL s		‡	‡	X	0	X	P	0	‡	010					Instruction format and states are as shown for RLC's. To form new Op-Code replace 000 of RLC's with shown code
RRC s		‡	‡	X	0	X	P	0	‡	001					
RR s		‡	‡	X	0	X	P	0	‡	011					
SLA s		‡	‡	X	0	X	P	0	‡	100					
SRA s		‡	‡	X	0	X	P	0	‡	101					
SRL s		‡	‡	X	0	X	P	0	‡	111					
RLD		‡	‡	X	0	X	P	0	•	11 101 101 01 101 111	ED 6F	2	5	18	Rotate digit left and right between the accumulator and location (HL). The content of the upper half of the accumulator is unaffected
RRD		‡	‡	X	0	X	P	0	•	11 101 101 01 100 111	ED 67	2	5	18	

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

Table 7.0-7

BIT SET, RESET AND TEST GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
BIT b, r	$Z \leftarrow \overline{r}_b$	X	‡	X	1	X	X	0	•	11 001 011 01 b r	CB	2	2	8	r: 000 B, 001 C, 010 D, 011 E, 100 H, 101 L, 111 A b: 000 0, 001 1, 010 2, 011 3, 100 4, 101 5, 110 6, 111 7
BIT b, (HL)	$Z \leftarrow \overline{(HL)}_b$	X	‡	X	1	X	X	0	•	11 001 011 01 b 110	CB	2	3	12	
BIT b, (IX+d)$_b$	$Z \leftarrow \overline{(IX+d)}_b$	X	‡	X	1	X	X	0	•	11 011 101 11 001 011 ← d → 01 b 110	DD CB	4	5	20	
BIT b, (IY+d)$_b$	$Z \leftarrow \overline{(IY+d)}_b$	X	‡	X	1	X	X	0	•	11 111 101 11 001 011 ← d → 01 b 110	FD CB	4	5	20	
SET b, r	$r_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 001 011 11 b r	CB	2	2	8	
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 001 011 11 b 110	CB	2	4	15	
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 011 101 11 001 011 ← d → 11 b 110	DD CB	4	6	23	
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 111 101 11 001 011 ← d → 11 b 110	FD CB	4	6	23	
RES b, s	$s_b \leftarrow 0$ s ≡ r, (HL), (IX+d), (IY+d)	•	•	X	•	X	•	•	•	10					To form new Op-Code replace 11 of SET b, s with 10. Flags and time states for SET instruction

Notes: The notation s_b indicates bit b (0 to 7) or location s.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

Table 7.0-8

JUMP GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	X	•	X	•	•	•	11 000 011 ← n → ← n →	C3	3	3	10	
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•	•	X	•	X	•	•	•	11 cc 010 ← n → ← n →		3	3	10	
JR e	PC ← PC + e	•	•	X	•	X	•	•	•	00 011 000 ← e-2 →	18	2	3	12	
JR C, e	If C = 0, continue	•	•	X	•	X	•	•	•	00 111 000 ← e-2 →	38	2	2	7	If condition not met
	If C = 1, PC ← PC+e											2	3	12	If condition is met
JR NC, e	If C = 1, continue	•	•	X	•	X	•	•	•	00 110 000 ← e-2 →	30	2	2	7	If condition not met
	If C = 0, PC ← PC+e											2	3	12	If condition is met
JR Z, e	If Z = 0 continue	•	•	X	•	X	•	•	•	00 101 000 ← e-2 →	28	2	2	7	If condition not met
	If Z = 1, PC ← PC+e											2	3	12	If condition is met
JR NZ, e	If Z = 1, continue	•	•	X	•	X	•	•	•	00 100 000 ← e-2 →	20	2	2	7	If condition not met
	If Z = 0, PC ← PC+e											2	3	12	If condition is met
JP (HL)	PC ← HL	•	•	X	•	X	•	•	•	11 101 001	E9	1	1	4	
JP (IX)	PC ← IX	•	•	X	•	X	•	•	•	11 011 101 11 101 001	DD E9	2	2	8	
JP (IY)	PC ← IY	•	•	X	•	X	•	•	•	11 111 101 11 101 001	FD E9	2	2	8	
DJNZ, e	B ← B-1 If B = 0, continue	•	•	X	•	X	•	•	•	00 010 000 ← e-2 →	10	2	2	8	If B = 0
	If B ≠ 0, PC ← PC+e											2	3	13	If B ≠ 0

cc	Condition
000	NZ non zero
001	Z zero
010	NC non carry
011	C carry
100	PO parity odd
101	PE parity even
110	P sign positive
111	M sign negative

Notes: e represents the extension in the relative addressing mode.

e is a signed two's complement number in the range <126, 129>

e-2 in the op-code provides an effective address of pc+e as PC is incremented by 2 prior to the addition of e.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

Table 7.0-9

CALL AND RETURN GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
CALL nn	(SP-1) ← PC_H (SP-2) ← PC_L PC ← nn	•	•	X	•	X	•	•	•	11 001 101 ← n → ← n →	CD	3	5	17	
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	•	•	X	•	X	•	•	•	11 cc 100 ← n → ← n →		3	3	10	If cc is false
												3	5	17	If cc is true
RET	PC_L ← (SP) PC_H ← (SP+1)	•	•	X	•	X	•	•	•	11 001 001	C9	1	3	10	
RET cc	If condition cc is false continue, otherwise same as RET	•	•	X	•	X	•	•	•	11 cc 000		1	1	5	If cc is false
												1	3	11	If cc is true
RETI	Return from interrupt	•	•	X	•	X	•	•	•	11 101 101 01 001 101	ED 4D	2	4	14	
RETN[1]	Return from non maskable interrupt	•	•	X	•	X	•	•	•	11 101 101 01 000 101	ED 45	2	4	14	
RST p	(SP-1) ← PC_H (SP-2) ← PC_L PC_H ← 0 PC_L ← p	•	•	X	•	X	•	•	•	11 t 111		1	3	11	

If cc is true

cc	Condition	
000	NZ	non zero
001	Z	zero
010	NC	non carry
011	C	carry
100	PO	parity odd
101	PE	parity even
110	P	sign positive
111	M	sign negative

t	p
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

[1] RETN loads IFF_2 → IFF_1

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

Z80 Device Family

Table 7.0-10

INPUT AND OUTPUT GROUP

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
IN A, (n)	A ← (n)	•	•	X	•	X	•	•	•	11 011 011 ← n →	DB	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	↕	↕	X	↕	X	P	0	•	11 101 101 01 r 000	ED	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	X	↕ ①	X	X	X	X	1	X	11 101 101 10 100 010	ED A2	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	1	X	11 101 101 10 110 010	ED B2	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	X	↕ ①	X	X	X	X	1	X	11 101 101 10 101 010	ED AA	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INDR	(HL) ← (C) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	X	1	X	11 101 101 10 111 010	ED BA	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUT (n), A	(n) ← A	•	•	X	•	X	•	•	•	11 010 011	D3	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
OUT (C), r	(C) ← r	•	•	X	•	X	•	•	•	11 101 101 01 r 001	ED	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTI	(C) ← (HL) B ← B - 1 HL ← HL + 1	X	↕ ①	X	X	X	X	1	X	11 101 101 10 100 011	ED A3	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTIR	(C) ← (HL) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	1	X	11 101 101 10 110 011	ED B3	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	X	↕ ①	X	X	X	X	1	X	11 101 101 10 101 011	ED AB	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	X	1	X	11 101 101 10 111 011	ED BB	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$

Notes: ① If the result of B - 1 is zero the Z flag is set, otherwise it is reset.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

Table 7.0-11

8.0 INTERRUPT RESPONSE

The prupose of an interrupt is to allow peripheral devices to suspend CPU operation in an orderly manner and force the CPU to start a peripheral service routine. Usually this service routine is involved with the exchange of data, or status and control information, between the CPU and the peripheral. Once the service routine is completed, the CPU returns to the operation from which it was interrupted.

INTERRUPT ENABLE – DISABLE

The Z80-CPU has two interrupt inputs, a software maskable interrupt and a non-maskable interrupt. The non-maskable interrupt ($\overline{NMI}$) can not be disabled by the programmer and it will be accepted whenever a peripheral device requests it. This interrupt is generally reserved for very important functions that must be serviced whenever they occur, such as an impending power failure. The maskable interrupt ($\overline{INT}$) can be selectively enabled or disabled by the programmer. This allows the programmer to disable the interrupt during periods where his program has timing constraints that do not allow it to be interrupted. In the Z80-CPU there is an enable flip flop (called IFF) that is set or reset by the programmer using the Enable Interrupt (EI) and Disable Interrupt (DI) instructions. When the IFF is reset, an interrupt can not be accepted by the CPU.

Actually, for purposes that will be subsequently explained, there are two enable flip flops, called IFF_1 and IFF_2.

IFF_1	IFF_2
Actually disables interrupts from being accepted.	Temporary storage location for IFF_1.

The state of IFF_1 is used to actually inhibit interrupts while IFF_2 is used as a temporary storage location for IFF_1. The purpose of storing the IFF_1 will be subsequently explained.

A reset to the CPU will force both IFF_1 and IFF_2 to the reset state so that interrupts are disabled. They can then be enabled by an EI instruction at any time by the programmer. When an EI instruction is executed, any pending interrupt request will not be accepted until after the instruction following EI has been executed. This single instruction delay is necessary for cases when the following instruction is a return instruction and interrupts must not be allowed until the return has been completed. The EI instructions sets both IFF_1 and IFF_2 to the enable state. When an interrupt is accepted by the CPU, both IFF_1 and IFF_2 are automatically reset, inhibiting further interrupts until the programmer wishes to issue a new EI instruction. Note that for all of the previous cases, IFF_1 and IFF_2 are always equal.

The purpose of IFF_2 is to save the status of IFF_1 when a non-maskable interrupt occurs. When a non-maskable interrupt is accepted, IFF_1 is reset to prevent further interrupts until reenabled by the programmer. Thus, after a non-maskable interrupt has been accepted maskable interrupts are disabled but the previous state of IFF_1 has been saved so that the complete state of the CPU just prior to the non-maskable interrupt can be restored at any time. When a Load Register A with Register I (LD A, I) instruction or a Load Register A with Register R (LD A, R) instruction is executed, the state of IFF_2 is copied into the parity flag where it can be tested or stored.

A second method of restoring the status of IFF_1 is thru the execution of a Return From Non-Maskable Interrupt (RETN) instruction. Since this instruction indicates that the non maskable interrupt service routine is complete, the contents of IFF_2 are now copied back into IFF_1, so that the status of IFF_1 just prior to the acceptance of the non-maskable interrupt will be restored automatically.

Figure 8.0-1 is a summary of the effect of different instructions on the two enable flip flops.

INTERRUPT ENABLE/DISABLE FLIP FLOPS

Action	IFF_1	IFF_2	
CPU Reset	0	0	
DI	0	0	
EI	1	1	
LD A, I	•	•	$IFF_2 \rightarrow$Parity flag
LD A, R	•	•	$IFF_2 \rightarrow$Parity flag
Accept NMI	0	•	
RETN	IFF_2	•	$IFF_2 \rightarrow IFF_1$
Accept INT	0	0	
RETI	•	•	

"•" indicates no change

FIGURE 8.0-1

CPU RESPONSE

Non-Maskable

A non-maskable interrupt will be accepted at all times by the CPU. When this occurs, the CPU ignores the next instruction that it fetches and instead does a restart to location 0066H. Thus, it behaves exactly as if it had received a restart instruction but, it is to a location that is not one of the 8 software restart locations. A restart is merely a call to a specific address in page 0 memory.

Maskable

The CPU can be programmed to respond to the maskable interrupt in any one of three possible modes.

Mode 0

This mode is identical to the 8080A interrupt response mode. With this mode, the interrupting device can place any instruction on the data bus and the CPU will execute it. Thus, the interrupting device provides the next instruction to be executed instead of the memory. Often this will be a restart instruction since the interrupting device only need supply a single byte instruction. Alternatively, any other instruction such as a 3 byte call to any location in memory could be executed.

The number of clock cycles necessary to execute this instruction is 2 more than the normal number for the instruction. This occurs since the CPU automatically adds 2 wait states to an interrupt response cycle to allow sufficient time to implement an external daisy chain for priority control. Section 4.0 illustrates the detailed timing for an interrupt response. After the application of RESET the CPU will automatically enter interrupt Mode 0.

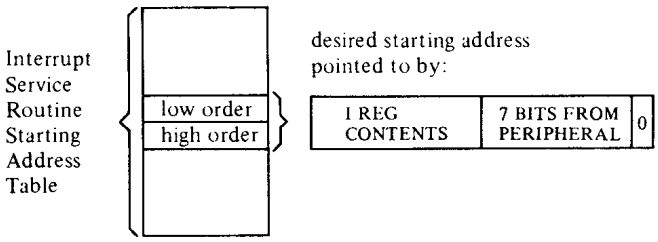
Mode 1

When this mode has been selected by the programmer, the CPU will respond to an interrupt by executing a restart to location 0038H. Thus the response is identical to that for a non maskable interrupt except that the call location is 0038H instead of 0066H. Another difference is that the number of cycles required to complete the restart instruction is 2 more than normal due to the two added wait states.

Mode 2

This mode is the most powerful interrupt response mode. With a single 8-bit byte from the user an indirect call can be made to any memory location.

With this mode the programmer maintains a table of 16 bit starting addresses for every interrupt service routine. This table may be located anywhere in memory. When an interrupt is accepted, a 16 bit pointer must be formed to obtain the desired interrupt service routine starting address from the table. The upper 8 bits of this pointer is formed from the contents of the I register. The I register must have been previously loaded with the desired value by the programmer, i.e. LD I, A. Note that a CPU reset clears the I register so that it is initialized to zero. The lower eight bits of the pointer must be supplied by the interrupting device. Actually, only 7 bits are required from the interrupting device as the least bit must be a zero. This is required since the pointer is used to get two adjacent bytes to from a complete 16 bit service routine starting address and the addresses must always start in even locations.



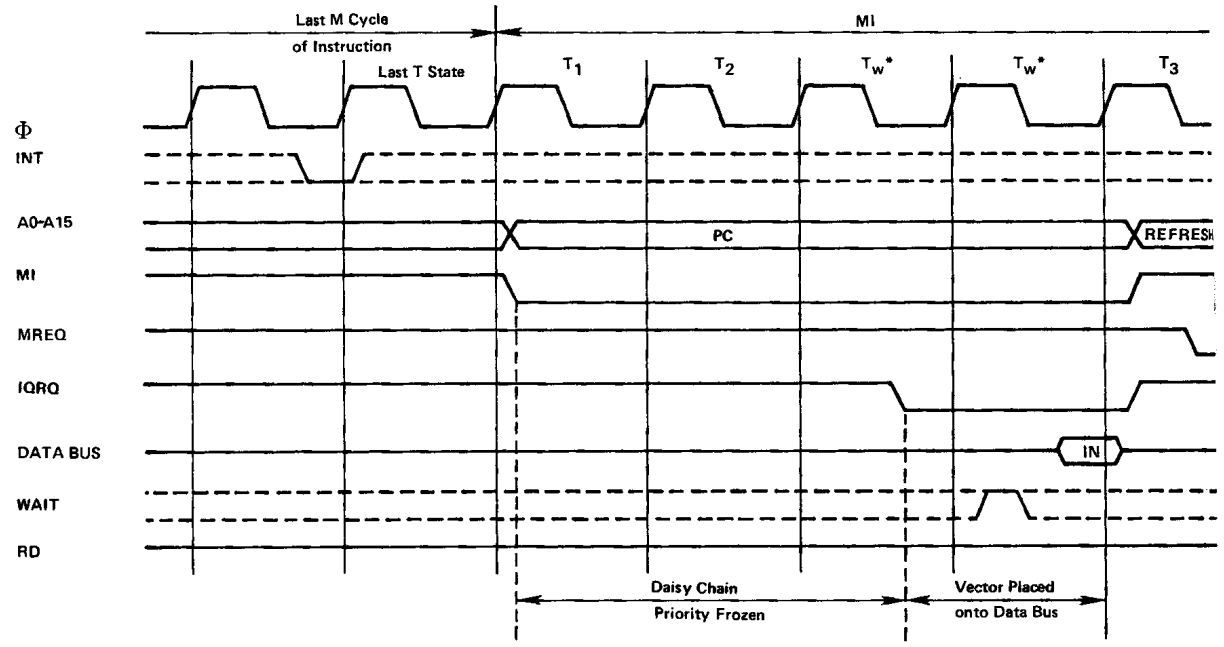
The first byte in the table is the least significant (low order) portion of the address. The programmer must obviously fill this table in with the desired addresses before any interrupts are to be accepted.

Note that this table can be changed at any time by the programmer (if it is stored in Read/Write Memory) to allow different peripherals to be serviced by different service routines.

Once the interrupting device supplies the lower portion of the pointer, the CPU automatically pushes the program counter onto the stack, obtains the starting address from the table and does a jump to this address. This mode of response requires 19 clock periods to complete (7 to fetch the lower 8 bits from the interrupting device, 6 to save the program counter, and 6 to obtain the jump address.)

Note that the Z80 peripheral devices all include a daisy chain priority interrupt structure that automatically supplies the programmed vector to the CPU during interrupt acknowledge. Refer to the Z80-PIO, Z80-SIO and Z80-CTC manuals for details.

INTERRUPT REQUEST/ACKNOWLEDGE CYCLE



Z80 INTERRUPT ACKNOWLEDGE SUMMARY

1) PERIPHERAL DEVICE REQUESTS INTERRUPT. Any device requesting and interrupt can pull the wired-or line $\overline{INT}$ low.

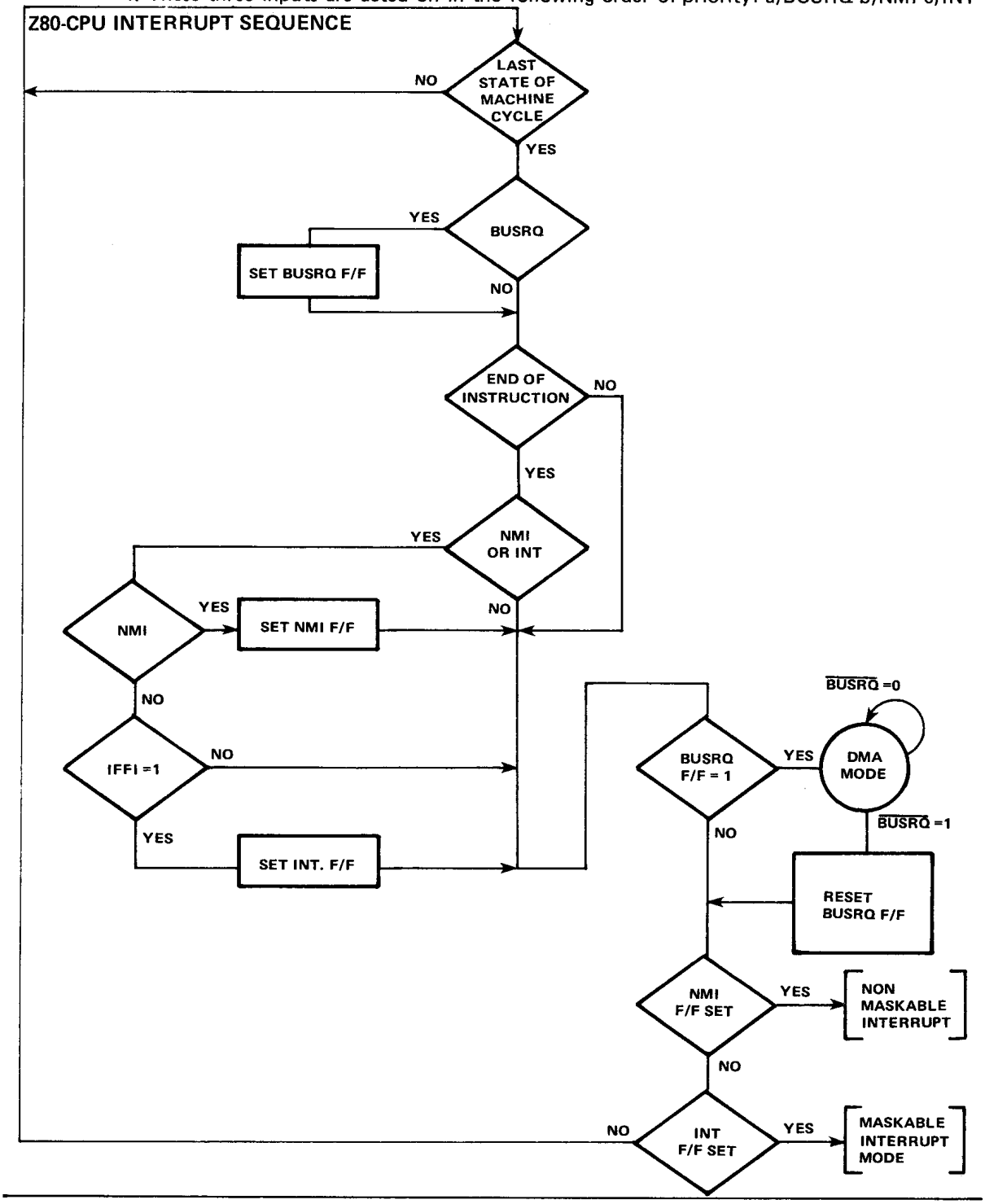
2) CPU ACKNOWLEDGES INTERRUPT. Priority status is frozen when $\overline{M1}$ goes low during the Interrupt Acknowledge sequence. Propagation delays down the IEI/IEO daisy chain must be settled out when $\overline{IORQ}$ goes low. If IEI is HIGH, an active Peripheral Device will place its Interrupt Vector on the Data Bus when $\overline{IORQ}$ goes low. That Peripheral then releases its hold on $\overline{INT}$ allowing interrupts from a higher priority device. Lower priority devices are inhibited from placing their Vector on the Data Bus or Interrupting because IEO is low on the active device.

3) INTERRUPT IS CLEARED. An active Peripheral device (IEI=1, IEO=0) monitors OP Code fetches for an RETI (ED 4D) instruction which tells the peripheral that its Interrupt Service Routine is over. The peripheral device then re-activates its internal Interrupt structure as well as raising its IEO line to enable lower priority devices.

INTERRELATIONSHIP OF $\overline{INT}$, $\overline{NMI}$, AND $\overline{BUSRQ}$

The following flow chart details the relationship of three control inputs to the Z80-CPU. Note the following from the flow chart.

1. $\overline{INT}$ and $\overline{NMI}$ are always acted on at the end of an instruction.
2. $\overline{BUSRQ}$ is acted on at the end of a machine cycle.
3. While the CPU is in the DMA MODE, it will not respond to active inputs on $\overline{INT}$ or $\overline{NMI}$.
4. These three inputs are acted on in the following order of priority: a)$\overline{BUSRQ}$ b)$\overline{NMI}$ c)$\overline{INT}$



(Page intentionally left blank)

9.0 HARDWARE IMPLEMENTATION EXAMPLES

This chapter is intended to serve as a basic introduction to implementing systems with the Z80-CPU.

MINIMUM SYSTEM

Figure 9.0-1 is a diagram of a very simple Z80 system. Any Z80 system must include the following five elements:

1) Five volt power supply
2) Oscillator
3) Memory devices
4) I/O circuits
5) CPU

MINIMUM Z80 COMPUTER SYSTEM

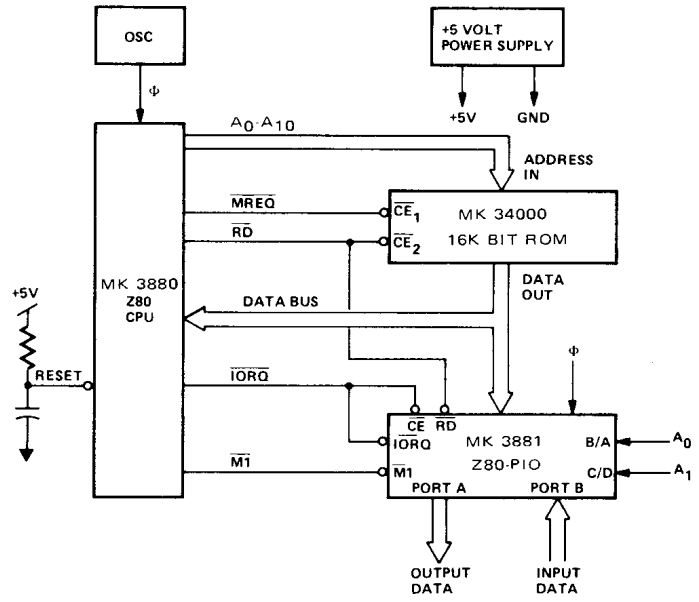


FIGURE 9.0-1

Since the Z80-CPU only requires a single 5 volt supply, most small systems can be implemented using only this single supply.

The oscillator can be very simple since the only requirement is that it be a 5 volt square wave. For systems not running at full speed, a simple RC oscillator can be used. When the CPU is operated near the highest possible frequency, a crystal oscillator is generally required because the system timing will not tolerate the drift or jitter that an RC network will generate. A crystal oscillator can be made from inverters and a few discrete components or monolithic circuits are widely available.

The external memory can be any mixture of standard RAM, ROM, or PROM. In this simple example we have shown a single 16K bit ROM (2K bytes) being utilized as the entire memory system. For this example we have assumed that the Z80 internal register configuration contains sufficient Read/Write storage so that external RAM memory is not required.

Every computer system requires I/O circuits to allow it to interface to the "real world." In this simple example it is assumed that the output is an 8 bit control vector and the input is an 8 bit status word. The input data could be gated onto the data bus using any standard tri-state driver while the output data could be latched with any type of standard TTL latch. For this example we have used a Z80-PIO for the I/O circuit. This single circuit attaches to the data bus as shown and provides the required 16 bits of TTL compatible I/O. (Refer to the Z80-PIO manual for details on the operation of this circuit.) Notice in this example that with only three LSI circuits, a simple oscillator and a single 5 volt power supply, a powerful computer has been implemented.

ADDING RAM

Most computer systems require some amount of external Read/Write memory for data storage and to implement a "stack". Figure 9.0-2 illustrates how 256 bytes of static memory can be added to the previous example. In this example the memory space is assumed to be organized as follows:

ROM & RAM IMPLEMENTATION EXAMPLE

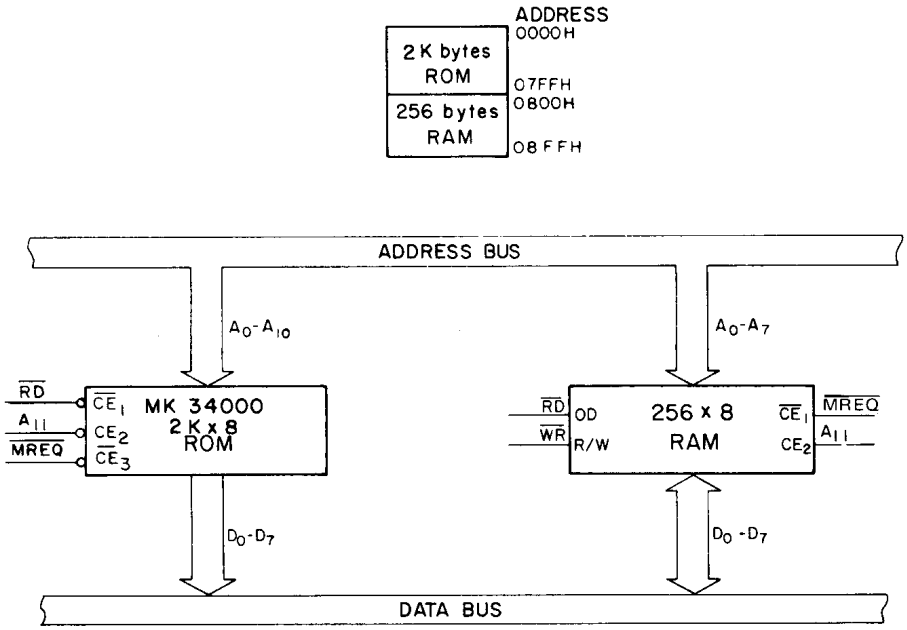


FIGURE 9.0-2

In this diagram the address space is described in hexidecimal notation. For this example, address bit A_{11} separates the ROM space from the RAM space so that it can be used for the chip select function. For larger amounts of external ROM or RAM, a simple TTL decoder will be required to form the chip selects.

MEMORY SPEED CONTROL

For many applications, it may be desirable to use slow memories to reduce costs. The $\overline{WAIT}$ line on the CPU allows the Z80 to operate with any speed memory. By referring back to section 4 you will notice that the memory access time requirements are most severe during the M1 cycle instruction fetch. All other memory accesses have an additional one half of a clock cycle to be completed. For this reason it may be desirable in some applications to add one wait state to the M1 cycle so that slower memories can be used. Figure 9.0-3 is an example of a simple circuit that will accomplish this task. This circuit can be changed to add a single wait state to any memory access as shown in Figure 9.0-4.

ADDING ONE WAIT STATE TO AN M1 CYCLE

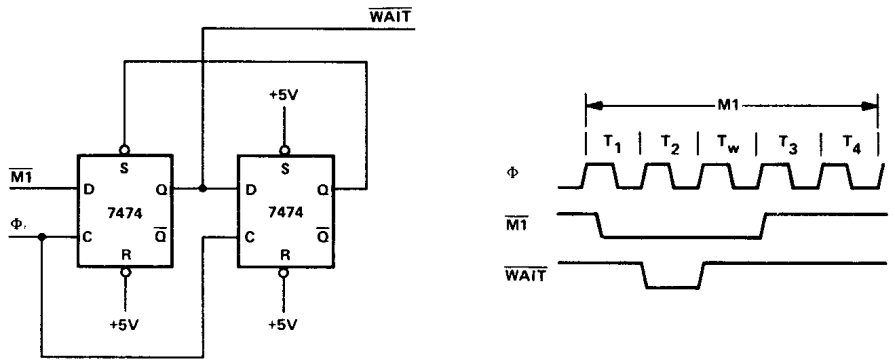


FIGURE 9.0-3

ADDING ONE WAIT STATE TO ANY MEMORY CYCLE

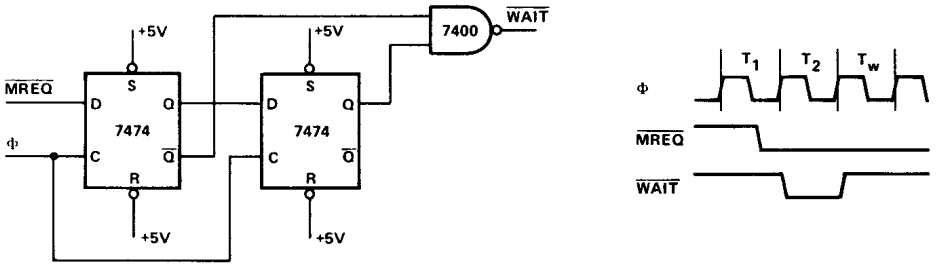


FIGURE 9.0-4

INTERFACING DYNAMIC MEMORIES

This section is intended only to serve as a brief introduction to interfacing dynamic memories. Each individual dynamic RAM has varying specifications that will require minor modifications to the description given here and no attempt will be made in this document to give details for any particular RAM.

Figure 9.0-5 illustrates the logic necessary to interface 8K bytes of dynamic RAM using 16-pin 4K dynamic memories. This Figure assumes that the RAM's are the only memory in the system so that A_{12} is used to select between the two pages of memory. During refresh time, all memories in the system must be read. The CPU provides the proper refresh address on lines A_0 through A_6. To add additional memory to the system it is necessary to only replace the two gates that operate on A_{12} with a decoder that operates on all required address bits. For larger systems, buffering for the address and data bus is also generally required.

An application note entitled "Z80 Interfacing Techniques for Dynamic RAM" is available from your MOSTEK representative which describes dynamic RAM design techniques.

INTERFACING DYNAMIC RAMS

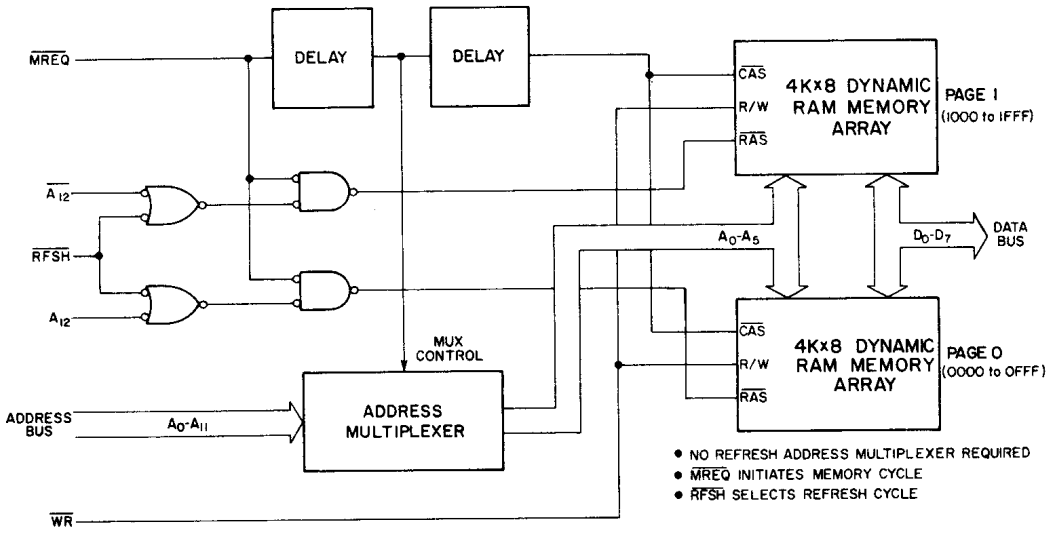


FIGURE 9.0-5

Z80 Device Family

Z80—CPU DESIGN CONSIDERATIONS: CLOCK CIRCUITRY

When using the Z80-CPU at less than its rated speed, the Clock Input (Φ) can be driven by a 7400 TTL gate with a resistor pull up (typically 330 ohms) to +5 Volts. Because of dynamic currents flowing into the Clock Input Pin, the rise time of the Clock Input waveform will be typically 60-80 nanoseconds. The resistor will eventually pull the clock input up to Vcc but with a slow rise time which will limit the maximum frequency of operation. Figure 9.0-6 shows a Clock Input driver which has an active pull-up and which will allow maximum frequency operation. The circuit is recommended for all but the most cost sensitive Z80 applications.

Z80 CPU CLOCK BUFFER CIRCUITRY

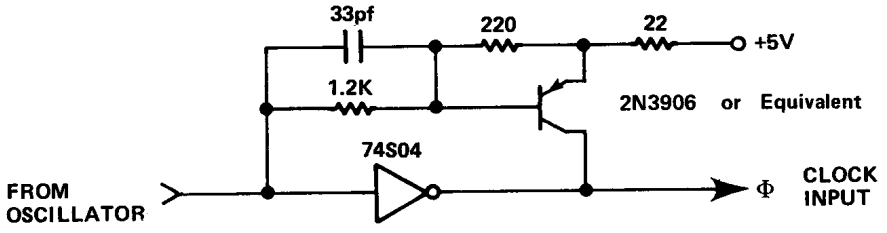


FIGURE 9.0-6

RESET CIRCUITRY

The Z80-CPU has the characteristic that if the $\overline{\text{RESET}}$ input goes low during T2 or T4 of a cycle that the MREQ signal will go to an indeterminate state for one T-State approximately 3 T-States later. If there are dynamic memories in the system this action could cause an aborted or short access of the dynamic RAM which could cause destruction of data within the RAM. If the contents of RAM are of no concern after RESET, then this characteristic is no problem as the CPU always resets properly. If RAM contents must be preserved, then the falling edge of the $\overline{\text{RESET}}$ input must be synchronized by the falling edge of $\overline{\text{M1}}$.

The circuitry of Figure 9.0-7 does this synchronization as well as providing a one-shot to limit the duration of the CPU $\overline{\text{RESET}}$ pulse. The CPU $\overline{\text{RESET}}$ signal must be a pulse even though the EXTERNAL RESET button is held closed to avoid suspending the CPU refresh of dynamic RAM for a time long enough to destroy data in the RAM.

MANUAL AND POWER–ON RESET CIRCUIT

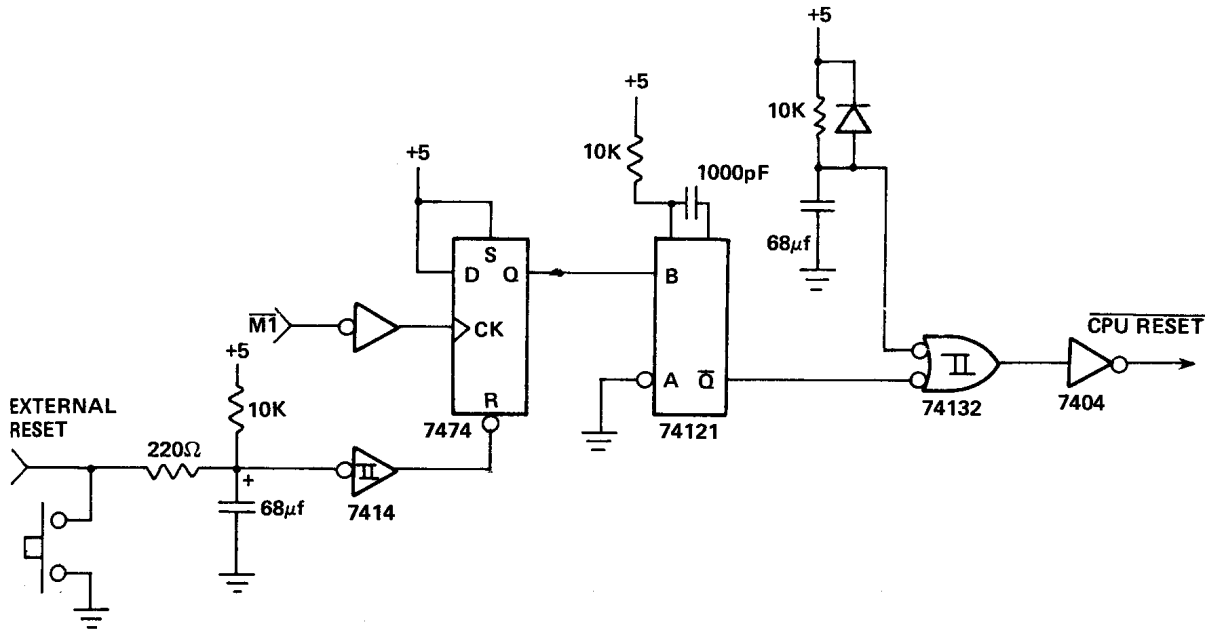


FIGURE 9.0-7

ADDRESS LATCHING

In order to guarantee proper operation of the Z80-CPU with dynamic RAMs the upper 4 bits of the address should be latched as shown in Figure 9.0-8. This action is required because the Z80-CPU does not guarantee that the Address Bus will hold valid before the rising edge of MREQ on an OP Code Fetch.

This action does not directly affect dynamic memories because they latch addresses internally. The problem comes from the address decoder which generates RAS. If the address lines which drive the decoder are allowed to change while $\overline{\text{MREQ}}$ is low, then a "glitch" can occur on the $\overline{\text{RAS}}$ line or lines, which may have the effect of destroying one row of data within the dynamic RAM.

ADDRESS LATCH

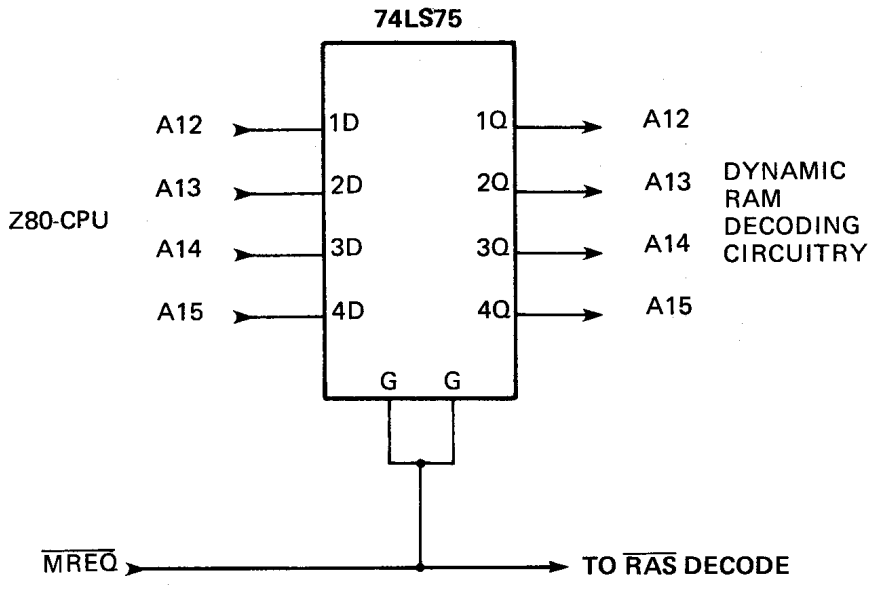


FIGURE 9.0-8

RAS TIMING WITH AND WITHOUT ADDRESS LATCH

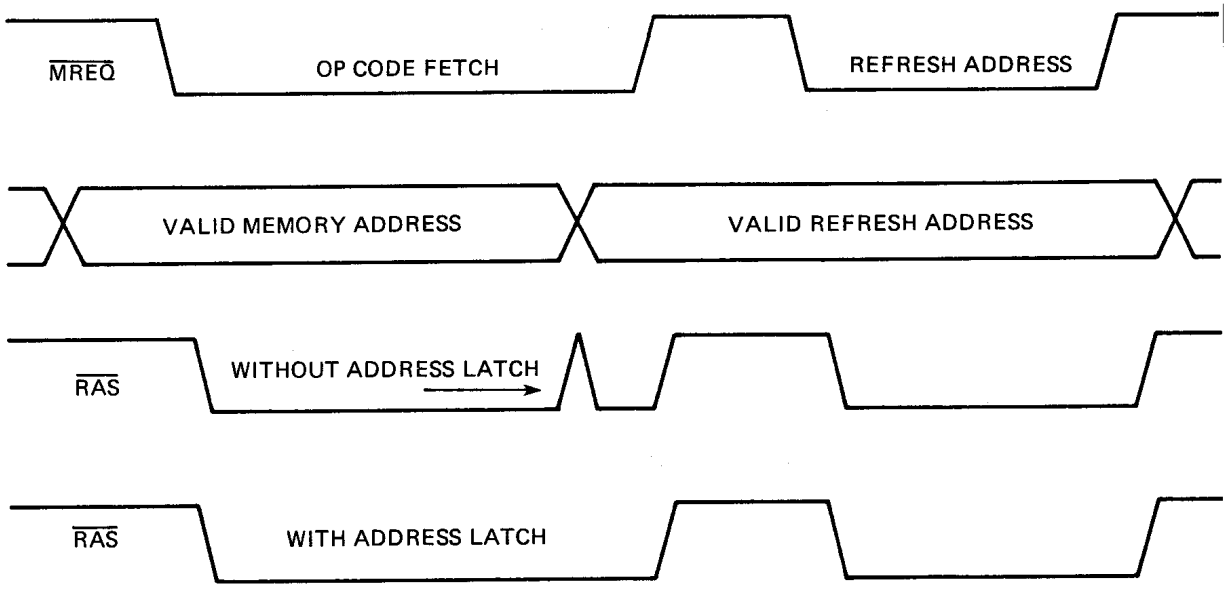


FIGURE 9.0-9

10.0 SOFTWARE IMPLEMENTATION EXAMPLES

10.1 Methods of Software Implementation

Several different approaches are possible in developing software for the Z80 (Figure 10.1) First of all, Assembly Language or a high level language may be used as the source language. These languages may then be translated into machine language on a commercial time sharing facility using a cross-assembler or cross-compiler or, in the case of assembly language, the translation can be accomplished on a Z80 Development System using a resident assembler. Finally, the resulting machine code can be debugged either on a time-sharing facility using a Z80 simulator or on a Z80 Development System which uses a Z80-CPU directly.

SOFTWARE GENERATION TECHNIQUES

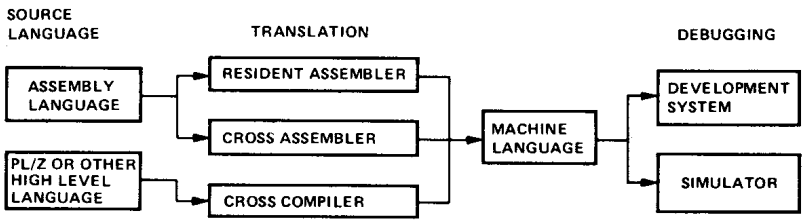


FIGURE 10.1

In selecting a source language, the primary factors to be considered are clarity and ease of programming vs. code efficiency. A high level language with its machine independent constraints is typically better for formulating and maintaining algorithms, but the resulting machine code is usually somewhat less efficient than what can be written directly in assembly language. These tradeoffs can often be balanced by combining high level language and assembly language routines, identifying those portions of a task which must be optimized and writing them as assembly language subroutines.

Deciding whether to use a resident or cross assembler is a matter of availability and short-term vs. long-term expense. While the initial expenditure for a development system is higher than that for a time-sharing terminal, the cost of an individual assembly using a resident assembler is negligible while the same operation on a time-sharing system is relatively expensive and in a short time this cost can equal the total cost of a development system.

Debugging on a development system vs. a simulator is also a matter of availability and expense combined with operational fidelity and flexibility. As with the assembly process, debugging is less expensive on a development system than on a simulator available through time-sharing. In addition, the fidelity of the operating environment is preserved through real-time execution on a Z80-CPU and by connecting the I/O and memory components which will actually be used in the production system. The only advantage to the use of a simulator is the range of criteria which may be selected for such debugging procedures as tracing and setting breakpoints. This flexibility exists because a software simulation can achieve any degree of complexity in its interpretation of machine instructions while development system procedures have hardware limitations such as the capacity of the real-time storage module, the number of breakpoint registers and the pin configuration of the CPU. Despite such hardware limitations, debugging on a development system is typically more productive than on a simulator because of the direct interaction that is possible between the programmer and the authentic execution of his program.

10.2 Software Features Offered by the Z80-CPU

The Z80 instruction set provides the user with a large and flexible repetoire of operations with which to formulate control of the Z80-CPU.

The primary, auxiliary and index registers can be used to hold the arguments of arithmetic and logical operations, or to form memory addresses, or as fast-access storage for frequently used data.

Information can be moved directly from register to register; from memory to memory; from memory to registers; or from registers to memory. In addition, register contents and register/memory contents can be exchanged without using temporary storage. In particular, the contents of primary and auxiliary registers can be completely exchanged by executing only two instructions. EX and EXX. This register exchange procedure can be used to separate the set of working registers between different logical procedures or to expand the set of available registers in a single procedure.

Storage and retrieval of data between pairs of registers and memory can be controlled on a last-in first-out basis through PUSH and POP instructions which utilize a special stack pointer register, SP. This stack register is available both to manipulate data and to automatically store and retrieve addresses for subroutine linkage. When a subroutine is called, for example, the address following the CALL instruction is placed on the top of the push-down stack pointed to by SP. When a subroutine returns to the calling routine, the address on the top of the stack is used to set the program counter for the address of the next instruction. The stack pointer is adjusted automatically to reflect the current "top" stack position during PUSH, POP, CALL and RET instructions. This stack mechanism allows pushdown data stacks and subroutine calls to be nested to any practical depth because the stack area can potentially be as large as memory space.

The sequence of instruction execution can be controlled by six different flags (carry, zero, sign, parity/overflow, add-subtract, half-carry) which reflect the results of arithmetic, logical, shift and compare instructions. After the execution of an instruction which sets a flag, that flag can be used to control a conditional jump or return instruction. These instructions provide logical control following the manipulation of single bit, eight-bit byte (or) sixteen-bit data quantities.

A full set of logical operations, including AND, OR, XOR (exclusive –OR), CPL (NOR) and NEG (two's complement) are available for Boolean operations between the accumulator and 1) all other eight-bit registers, 2) memory locations or 3) immediate operands.

In addition, a full set of arithmetic and logical shifts in both directions are available which operate on the contents of all eight-bit primary registers or directly on any memory location. The carry flag can be included or simply set by these shift instructions to provide both the testing of shift results and to link register/register or register/memory shift operations.

10.3 Examples of Use of Special Z80 Instructions

A. Let us assume that a string of data in memory starting at location "DATA" is to be moved into another area of memory starting at location "BUFFER" and that the string length is 737 bytes. This operation can be accomplished as follows:

```
LD      HL, DATA     ;START ADDRESS OF DATA STRING
LD      DE, BUFFER   ;START ADDRESS OF TARGET BUFFER
LD      BC, 737      ;LENGTH OF DATA STRING
LDIR                 ;MOVE STRING – TRANSFER MEMORY
                     ;POINTED TO BY HL INTO MEMORY
                     ;LOCATION POINTED TO BY DE INCREMENT
                     ;HL AND DE, DECREMENT BC PROCESS
                     ;UNTIL BC=0.
```

11 bytes are required for this operation and each byte of data is moved in 21 clock cycles.

B. Let's assume that a string in memory starting at location "DATA" is to be moved into another area of memory starting at location "BUFFER" until an ASCII $ character (used as string delimiter) is found. Let's also assume that the maximum string length is 132 characters. The operation can be performed as follows:

```
        LD      HL, DATA    ;STARTING ADDRESS OF DATA STRING
        LD      DE, BUFFER  ;STARTING ADDRESS OF TARGET BUFFER
        LD      BC, 132     ;MAXIMUM STRING LENGTH
        LD      A, '$'      ;STRING DELIMITER CODE
LOOP:   CP      (HL)        ;COMPARE MEMORY CONTENTS WITH DE-
                            ;LIMITER
        JR      Z, END-$    ;GO TO END IF CHARACTERS EQUAL
        LDI                 ;MOVE CHARACTER (HL) TO (DE)
                            ;INCREMENT HL AND DE, DECREMENT BC
        JP      PE,LOOP     ;GO TO "LOOP" IF MORE CHARACTERS
END:                        ;OTHERWISE, FALL THROUGH
                            ;NOTE: P/V FLAG IS USED
                            ;TO INDICATE THAT REGISTER BC WAS
                            ;DECREMENTED TO ZERO.
```

19 bytes are required for this operation.

C. Let us assume that a 16-digit decimal number represented in packed BCD format (two BCD digits/byte) has to be shifted as shown in the Figure 10.2 in order to mechanize BCD multiplication or division. The operation can be accomplished as follows:

```
        LD      HL, DATA    ;ADDRESS OF FIRST BYTE
        LD      B, COUNT    ;SHIFT COUNT
        XOR     A           ;CLEAR ACCUMULATOR
ROTAT:  RLD                 ;ROTATE LEFT LOW ORDER DIGIT IN ACC
                            ;WITH DIGITS IN (HL)
        INC     HL          ;ADVANCE MEMORY POINTER
        DJNZ    ROTAT-$     ;DECREMENT B AND GO TO ROTAT IF
                            ;B IS NOT ZERO, OTHERWISE FALL THROUGH
```


BCD DATA SHIFTING

11 bytes are required for this operation.

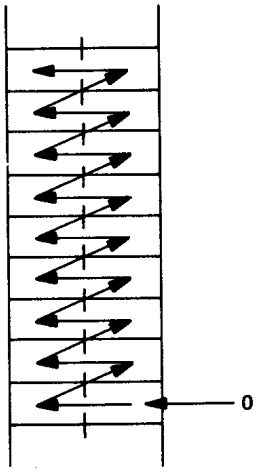


FIGURE 10.2

11 bytes are required for this operation.

D. Let us assume that one number is to be subtracted from another and a) that they are both in packed BCD format, b) that they are of equal but varying length, and c) that the result is to be stored in the location of the minuend. The operation can be accomplished as follows:

```
        LD      HL, ARG1        ;ADDRESS OF MINUEND
        LD      DE, ARG2        ;ADDRESS OF SUBTRAHEND
        LD      B, LENGTH       ;LENGTH OF TWO ARGUMENTS
        AND     A               ;CLEAR CARRY FLAG
SUBDEC: LD      A, (DE)         ;SUBTRAHEND TO ACC
        SBC     A, (HL)         ;SUBTRACT (HL) FROM ACC
        DAA                     ;ADJUST RESULT TO DECIMAL CODED VALUE
        LD      (HL), A         ;STORE RESULT
        INC     HL              ;ADVANCE MEMORY POINTERS
        INC     DE
        DJNZ    SUBDEC-$        ;DECREMENT B AND GO TO "SUBDEC" IF B
                                ;NOT ZERO, OTHERWISE FALL THROUGH
```

17 bytes are required for this operation.

10.4 Examples of Programming Tasks

A. The following program sorts an array of numbers each in the range <0,255> into ascending order using a standard exchange sorting algorithm.

```
01/22/76     11:14:37                BUBBLE LISTING
LOC    OBJ CODE  STMT  SOURCE STATEMENT

                1     ;  *** STANDARD EXCHANGE (BUBBLE) SORT ROUTINE***
                2     ;
                3     ;  AT ENTRY: HL CONTAINS ADDRESS OF DATA
                4     ;             C CONTAINS NUMBER OF ELEMENTS TO BE SORTED
                5     ;             (1<C<256)
                6     ;
                7     ;  AT EXIT: DATA SORTED IN ASCENDING ORDER
                8     ;
                9     ;  USE OF REGISTERS
                10    ;
                11    ;  REGISTER   CONTENTS
                12    ;
                13    ;  A          TEMPORARY STORAGE FOR CALCULATIONS
                14    ;  B          COUNTER FOR DATA ARRAY
                15    ;  C          LENGTH OF DATA ARRAY
                16    ;  D          FIRST ELEMENT IN COMPARISON
                17    ;  E          SECOND ELEMENT IN COMPARISON
                18    ;  H          FLAG TO INDICATE EXCHANGE
                19    ;  L          UNUSED
                20    ;  IX         POINTER INTO DATA ARRAY
                21    ;  IY         UNUSED
                22    ;
```

```
LOC   OBJ CODE   STMT  SOURCE STATMENT
0000  222600      23   SORT:   LD    (DATA), HL   ;SAVE DATA ADDRESS
0003  CB84        24   LOOP:   RES   FLAG, H      ;INITIALIZE EXCHANGE FLAG
0005  41          25           LD    B,C          ;INITIALIZE LENGTH COUNTER
0006  05          26           DEC   B            ;ADJUST FOR TESTING
0007  DD2A2600    27           LD    IX, (DATA)   ;INITIALIZE ARRAY POINTER
000B  DD7E00      28   NEXT:   LD    A,(IX+0)     ;FIRST ELEMENT IN COMPARISON
000E  57          29           LD    D, A         ;TEMPORARY STORAGE FOR ELEMENT
000F  DD5E01      30           LD    E, (IX+1)    ;SECOND ELEMENT IN COMPARISON
0012  93          31           SUB   E            ;COMPARISON FIRST TO SECOND
0013  3008        32           JR    NC, NOEX-$   ;IF FIRST> SECOND, NO JUMP
0015  DD7300      33           LD    (IX), E      ;EXCHANGE ARRAY ELEMENTS
0018  DD7201      34           LD    (IX+1), D
001B  CBC4        35           SET   FLAG H       ;RECORD EXCHANGE OCCURRED
001D  DD23        36   NOEX:   INC   IX           ;POINT TO NEXT DATA ELEMENT
001F  10EA        37           DJNZ  NEXT-$       ;COUNT NUMBER OF COMPARISONS
                                                  ;REPEAT IF MORE DATA PAIRS
0021  CB44        39           BIT   FLAG, H      ;DETERMINE IF EXCHANGE OCCURRED
0023  20DE        40           JR    NZ, LOOP-$   ;CONTINUE IF DATA UNSORTED
0025  C9          41           RET                ;OTHERWISE, EXIT
                  42   ;
0026              43   FLAG:   EQU   0            ;DESIGNATION OF FLAG BIT
0026              44   DATA:   DEFS  2            ;STORAGE FOR DATA ADDRESS
                  45           END
```

B. The following program multiplies two unsigned 16-bit integers and leaves the result in the HL register pair.

```
01/22/76    11:32:36          MULTIPLY LISTING
LOC   OBJ CODE   STMT  SOURCE STATEMENT

0000              1   MULT:;  UNSIGNED SIXTEEN BIT INTEGER MULTIPLY.
                  2   ;       ON ENTRANCE: MULTIPLIER IN HL.
                  3   ;                    MULTIPLICAND IN DE.
                  4   ;
                  5   ;       ON EXIT: RESULT IN HL.
                  6   ;
                  7   ;       REGISTERS USES:
                  8   ;
                  9   ;
                 10   ;       H   HIGH ORDER PARTIAL RESULT
                 11   ;       L   LOW ORDER PARTIAL RESULT
                 12   ;       D   HIGH ORDER MULTIPLICAND
                 13   ;       E   LOW ORDER MULTIPLICAND
                 14   ;       B   COUNTER FOR NUMBER OF SHIFTS
                 15   ;       C   HIGH ORDER BITS OF MULTIPLIER
                 16   ;       A   LOW ORDER BITS OF MULTIPLIER
                 17   ;
0000  0610       18           LD    B, 16;       NUMBER OF BITS--INITIALIZE
0002  4A         19           LD    C,D;         MOVE MULTIPLIER
0003  7B         20           LD    A,E;
0004  EB         21           EX    DE,HL;       MOVE MULTIPLICAND
0005  210000     22           LD    HL,0;        CLEAR PARTIAL RESULT
0008  CB39       23   MLOOP:  SRL   C;          SHIFT MULTIPLIER RIGHT
000A  1F         24           RR    A;           LEAST SIGNIFICANT BIT IS
                                                 IN CARRY.
000B  3001       26           JR    NC, NOADD-$  IF NO CARRY' SKIP THE ADD.
```

```
LOC   OBJ CODE   STMT  SOURCE STATMENT
000D  19         27            ADD HL, DE;          ELSE ADD MULTIPLICAND TO
                                                    PARTIAL RESULT.
000E  EB         29    NOADD:  EX  DE,HL;           SHIFT MULTIPLICANT LEFT
000F  29         30            ADD HL,HL;           BY MULTIPLYING IT BY TWO.
0010  EB         31            EX  DE,HL;
0011  10F5       32          DJNZ  MLOOP-$;         REPEAT UNTIL NO MORE BITS.
0013  C9         33            RET;
                  34            END;
```

11.0 ELECTRICAL SPECIFICATIONS
ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... Specified Operating Range

Storage Temperature ..... –65°C to +150°C

Voltage on Any Pin with Respect to Ground ..... –0.3V to +7V

Power Dissipation 1.5W

D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = 5V ± 5% unless otherwise specified

SYMBOL	PARAMETER	MIN.	TYP.	MAX.	UNIT	TEST CONDITION
V_{ILC}	Clock Input Low Voltage	–0.3		0.8	V	
V_{IHC}	Clock Input High Voltage	Vcc-.6		Vcc+.3	V	
V_{IL}	Input Low Voltage	–0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = –250 μA
I_{CC}	Power Supply Current			150*	mA	
I_{LI}	Input Leakage Current			10	μA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	V_{OUT} = 2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float			–10	μA	V_{OUT} = 0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \leqslant V_{IN} \leqslant V_{CC}$

*200mA for -4, -10 or -20 devices

CAPACITANCE

T_A = 25°C, f = 1MHz unmeasured pins returned to ground

SYMBOL	PARAMETER	MAX.	UNIT
$C\Phi$	Clock Capacitance	35	pF
C_{IN}	Input Capacitance	5	pF
C_{OUT}	Output Capacitance	10	pF

*Comment

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

A C CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5V ± 5%, Unless Otherwise Noted

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITION
Φ	t_c	Clock Period	.4	[12]	μsec	
	$t_{w(\Phi H)}$	Clock Pulse Width, Clock High	180	(D)	nsec	
	$t_{w(\Phi L)}$	Clock Pulse Width, Clock Low	180	2000	nsec	
	$t_{r,f}$	Clock Rise and Fall Time		30	nsec	
A_{0-15}	$t_{D(AD)}$	Address Output Delay		145	nsec	
	$t_{F(AD)}$	Delay to Float		110	nsec	
	t_{acm}	Address Stable Prior to $\overline{MREQ}$ (Memory Cycle)	[1]		nsec	C_L = 50pF
	t_{aci}	Address Stable Prior to $\overline{IORQ}$, $\overline{RD}$ or $\overline{WR}$ (I/O Cycle)	[2]		nsec	
	t_{ca}	Address Stable From $\overline{RD}$, $\overline{WR}$, $\overline{IORQ}$ or $\overline{MREQ}$	[3]		nsec	Except T3-M1
	t_{caf}	Address Stable From $\overline{RD}$ or $\overline{WR}$ During Float	[4]		nsec	
D_{0-7}	$t_{D(D)}$	Data Output Delay		230	nsec	
	$t_{F(D)}$	Delay to Float During Write Cycle		90	nsec	
	$t_{S\Phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	50		nsec	
	$t_{S\overline{\Phi}(D)}$	Data Setup Time to Falling Edge at Clock During M2 to M5	60		nsec	C_L = 50pF
	t_{dcm}	Data Stable Prior to $\overline{WR}$ (Memory Cycle)	[5]		nsec	
	t_{dci}	Data Stable Prior to $\overline{WR}$ (I/O Cycle)	[6]		nsec	
	t_{cdf}	Data Stable From $\overline{WR}$	[7]		nsec	
	t_H	Input Hold Time	0		nsec	
$\overline{MREQ}$	$t_{DL\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ Low		100	nsec	
	$t_{DH\Phi(MR)}$	$\overline{MREQ}$ Delay From Rising Edge of Clock, $\overline{MREQ}$ High		100	nsec	
	$t_{DH\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ High		100	nsec	C_L = 50 pF
	$t_{w(\overline{MRL})}$	Pulse Width, $\overline{MREQ}$ Low	[8]		nsec	
	$t_{w(\overline{MRH})}$	Pulse Width, $\overline{MREQ}$ High	[9]		nsec	
$\overline{IORQ}$	$t_{DL\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ Low		90	nsec	
	$t_{DL\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ Low		110	nsec	C_L = 50 pF
	$t_{DH\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ High		100	nsec	
	$t_{DH\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ High		110	nsec	
$\overline{RD}$	$t_{DL\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ Low		100	nsec	
	$t_{DL\overline{\Phi}(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ Low		130	nsec	C_L = 50pF
	$t_{DH\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ High		100	nsec	
	$t_{DH\overline{\Phi}(BD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ High		110	nsec	
$\overline{WR}$	$t_{DL\Phi(WR)}$	$\overline{WR}$ Delay From Rising Edge of Clock, $\overline{WR}$ Low		80	nsec	
	$t_{DL\overline{\Phi}(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock $\overline{WR}$ Low		90	nsec	C_L = 50pF
	$t_{DH\Phi(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ High		100	nsec	
	$t_{w(\overline{WRL})}$	Pulse Width, $\overline{WR}$ Low	[10]		nsec	

NOTES:

A Data should be enabled onto the CPU data bus when RD is active. During interrupt acknowledge data should be enabled when $\overline{M1}$ and $\overline{IORQ}$ are both active.

B The $\overline{RESET}$ signal must be active for a minimum of 3 clock cycles.

cont'd on page 81

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
	$t_{DL(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock $\overline{M1}$ Low		130	nsec	C_L = 50pF
$\overline{M1}$	$t_{DH(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock $\overline{M1}$ High		130	nsec	
$\overline{RFSH}$	$t_{DL(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ Low		180	nsec	C_L = 30pF
	$t_{DH(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ High		150	nsec	
$\overline{WAIT}$	$t_{S(WT)}$	$\overline{WAIT}$ Setup Time to Falling Edge of Clock	70		nsec	
$\overline{HALT}$	$t_{D(HT)}$	$\overline{HALT}$ Delay Time From Falling Edge of Clock		300	nsec	C_L = 50pF
$\overline{INT}$	$t_{s(IT)}$	$\overline{INT}$ Setup Time to Rising Edge of Clock	80		nsec	
$\overline{NMI}$	$t_{w(\overline{NML})}$	Pulse Width, $\overline{NMI}$ Low	80		nsec	
$\overline{BUSRQ}$	$t_{s(BQ)}$	$\overline{BUSRQ}$ Setup Time to Rising Edge of Clock	80		nsec	
$\overline{BUSAK}$	$t_{DL(BA)}$	$\overline{BUSAK}$ Delay From Rising Edge of Clock, $\overline{BUSAK}$ Low		120	nsec	C_L = 50 pF
	$t_{DH(BA)}$	$\overline{BUSAK}$ Delay From Falling Edge of Clock, $\overline{BUSAK}$ High		110	nsec	
$\overline{RESET}$	$t_{s(RS)}$	$\overline{RESET}$ Setup Time to Rising Edge of Clock	90		nsec	
	$t_{F(C)}$	Delay to/from Float ($\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ and $\overline{WR}$)		100	nsec	
	t_{mr}	$\overline{M1}$ Stable Prior to IORQ (Interrupt Ack.)	[11]		nsec	

[1] $t_{acm} = t_w(\Phi H) + t_f - 75$

[2] $t_{aci} = t_c - 80$

[3] $t_{ca} = t_w(\Phi L) + t_r - 40$

[4] $t_{caf} = t_w(\Phi L) + t_r - 60$

[5] $t_{dcm} = t_c - 210$

[6] $t_{dci} = t_w(\Phi L) + t_r - 210$

[7] $t_{cdf} = t_w(\Phi L) + t_r - 80$

[8] $t_{w(\overline{MRL})} = t_c - 40$

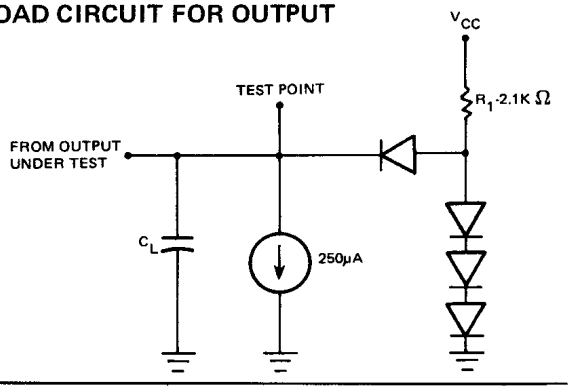
[9] $t_{w(\overline{MRH})} = t_w(\Phi H) + t_f - 30$

[10] $t_{w(\overline{WR})} = t_c - 40$

[11] $t_{mr} = 2t_c + t_w(\Phi H) + t_f - 80$

[12] $t_c = t_w(\Phi H) + t_w(\Phi L) + t_r + t_f$

LOAD CIRCUIT FOR OUTPUT



NOTES (Cont'd.)

C. Output Delay vs. Load Capacitance
T_A = 70°C V_{CC} = 5V±5%
Add 10 nsec delay for each 50pF increase in load up to a maximum of 200pF for the data bus and 100pF for address and control lines.

D. Although static by design, testing guarantees $t_w(\Phi H)$ of 200 μsec maximum.

A. C. CHARACTERISTICS T_A = 0°C to 70°C, Vcc = +5V ±5%, Unless Otherwise Noted

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITION
Φ	t_c	Clock Period	.25	[12]	μsec	
	$t_{w(\Phi H)}$	Clock Pulse Width, Clock High	110	(D)	nsec	
	$t_{w(\Phi L)}$	Clock Pulse Width, Clock Low	110	2000	nsec	
	$t_{r,f}$	Clock Rise and Fall Time		30	nsec	
A_{0-15}	$t_{D(AD)}$	Address Output Delay		110	nsec	
	$t_{F(AD)}$	Delay to Float		90	nsec	
	t_{acm}	Address Stable Prior to $\overline{MREQ}$ (Memory Cycle)	[1]		nsec	C_L = 50pF
	t_{aci}	Address Stable Prior to $\overline{IORQ}$, $\overline{RD}$ or $\overline{WR}$ (I/O Cycle)	[2]		nsec	
	t_{ca}	Address Stable From $\overline{RD}$, $\overline{WR}$, $\overline{IORQ}$ or $\overline{MREQ}$	[3]		nsec	Except T3.M1
	t_{caf}	Address Stable From $\overline{RD}$ or $\overline{WR}$ During Float	[4]		nsec	
D_{0-7}	$t_{D(D)}$	Data Output Delay		150	nsec	
	$t_{F(D)}$	Delay to Float During Write Cycle		90	nsec	
	$t_{S\Phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	50		nsec	
	$t_{S\overline{\Phi}(D)}$	Data Setup Time to Falling Edge at Clock During M2 to M5	60		nsec	C_L = 50pF
	t_{dcm}	Data Stable Prior to $\overline{WR}$ (Memory Cycle)	[5]		nsec	
	t_{dci}	Data Stable Prior to $\overline{WR}$ (I/O Cycle)	[6]		nsec	
	t_{cdf}	Data Stable From $\overline{WR}$	[7]		nsec	
	t_H	Input Hold Time	0		nsec	
$\overline{MREQ}$	$t_{DL\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ Low	20	85	nsec	
	$t_{DH\Phi(MR)}$	$\overline{MREQ}$ Delay From Rising Edge of Clock, $\overline{MREQ}$ High		85	nsec	
	$t_{DH\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ High		85	nsec	C_L = 50pF
	$t_{w(\overline{MRL})}$	Pulse Width, $\overline{MREQ}$ Low	[8]		nsec	
	$t_{w(\overline{MRH})}$	Pulse Width, $\overline{MREQ}$ High	[9]		nsec	
$\overline{IORQ}$	$t_{DL\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ Low		75	nsec	
	$t_{DL\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ Low		85	nsec	C_L = 50pF
	$t_{DH\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ High		85	nsec	
	$t_{DH\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ High		85	nsec	
$\overline{RD}$	$t_{DL\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ Low		85	nsec	
	$t_{DL\overline{\Phi}(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ Low		95	nsec	C_L = 50pF
	$t_{DH\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ High		85	nsec	
	$t_{DH\overline{\Phi}(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ High		85	nsec	
$\overline{WR}$	$t_{DL\Phi(WR)}$	$\overline{WR}$ Delay From Rising Edge of Clock, $\overline{WR}$ Low		65	nsec	
	$t_{DL\overline{\Phi}(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ Low		80	nsec	C_L = 50pF
	$t_{DH\Phi(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ High		80	nsec	
	$t_{w(\overline{WR}L)}$	Pulse Width, $\overline{WR}$ Low	[10]		nsec	

NOTES:

A Data should be enabled onto the CPU data bus when $\overline{RD}$ is active. During interrupt acknowledge data should be enabled when M1 and $\overline{IORQ}$ are both active.

B The $\overline{RESET}$ signal must be active for a minimum of 3 clock cycles.

(Cont'd. on page 83)

SIGNAL	SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITION
$\overline{M1}$	$t_{DL(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock $\overline{M1}$ Low		100	nsec	C_L = 50pF
	$t_{DH(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock, $\overline{M1}$ High		100	nsec	
$\overline{RFSH}$	$t_{DL(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ Low		130	nsec	C_L = 50pF
	$t_{DH(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock $\overline{RFSH}$ High		120	nsec	
$\overline{WAIT}$	$t_{S(WT)}$	$\overline{WAIT}$ Setup Time to Falling Edge of Clock	70		nsec	
$\overline{HALT}$	$t_{D(HT)}$	$\overline{HALT}$ Delay Time From Falling Edge of Clock		300	nsec	C_L = 50pF
$\overline{INT}$	$t_{s(IT)}$	$\overline{INT}$ Setup Time to Rising Edge of Clock	80		nsec	
$\overline{NMI}$	$t_{w(\overline{NML})}$	Pulse Width, $\overline{NMI}$ Low	80		nsec	
$\overline{BUSRQ}$	$t_{s(BQ)}$	$\overline{BUSRQ}$ Setup Time to Rising Edge of Clock	50		nsec	
$\overline{BUSAK}$	$t_{DL(BA)}$	$\overline{BUSAK}$ Delay From Rising Edge of Clock, $\overline{BUSAK}$ Low		100	nsec	C_L = 50pF
	$t_{DH(BA)}$	$\overline{BUSAK}$ Delay From Falling Edge of Clock, $\overline{BUSAK}$ High		100	nsec	
$\overline{RESET}$	$t_{s(RS)}$	$\overline{RESET}$ Setup Time to Rising Edge of Clock	60		nsec	
	$t_{F(C)}$	Delay to/From Float ($\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ and $\overline{WR}$)		80	nsec	
	t_{mr}	$\overline{M1}$ Stable Prior to $\overline{IORQ}$ (Interrupt Ack.)	[11]		nsec	

[1] $t_{acm} = t_w(\Phi H) + t_f - 65$

[2] $t_{aci} = t_c - 70$

[3] $t_{ca} = t_w(\Phi L) + t_r - 50$

[4] $t_{caf} = t_w(\Phi L) + t_r - 45$

[5] $t_{dcm} = t_c - 170$

[6] $t_{dci} = t_w(\Phi L) + t_r - 170$

[7] $t_{cdf} = t_w(\Phi L) + t_r - 70$

[8] $t_{w(\overline{MRL})} = t_c - 30$

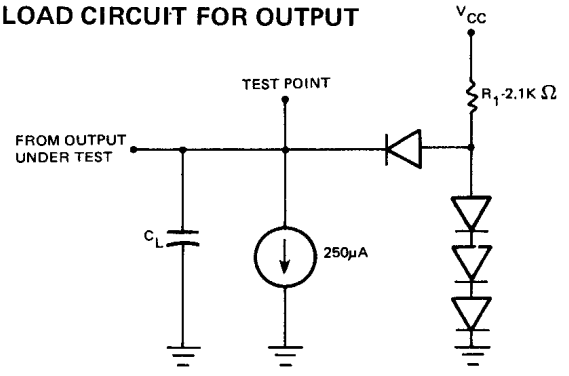
[9] $t_{w(\overline{MRH})} = t_w(\Phi H) + t_f - 20$

[10] $t_{w(\overline{WR})} = t_c - 30$

[11] $t_{mr} = 2t_c + t_w(\Phi H) + t_f - 65$

[12] $t_c = t_w(\Phi H) + t_w(\Phi L) + t_r + t_f$

LOAD CIRCUIT FOR OUTPUT



NOTES (Cont'd.)

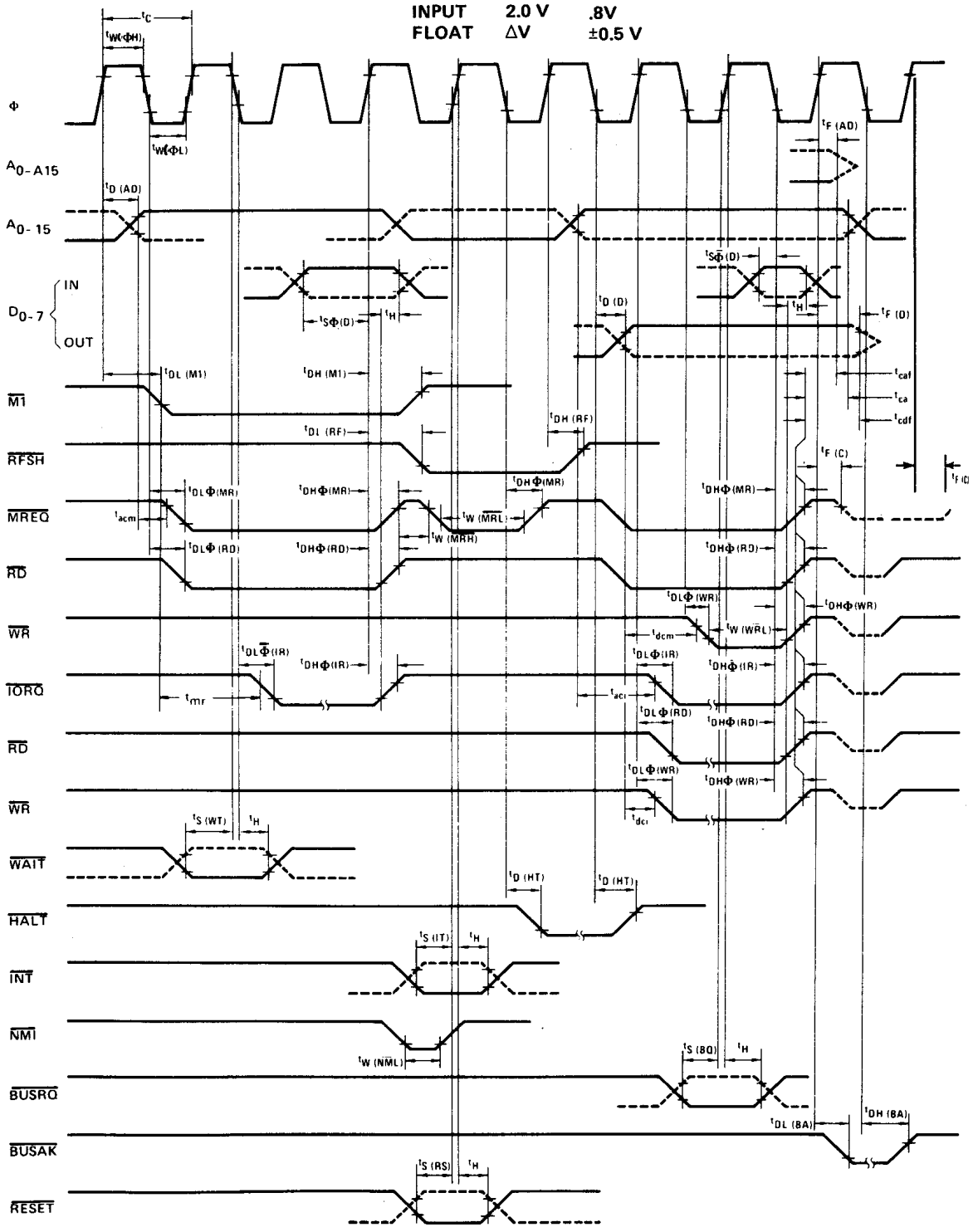
C. Output Delay vs. Load Capacitance
$T_A = 70°C$ $V_{CC} = 5V \pm 5\%$
Add 10 nsec delay for each 50pF increase in load up to a maximum of 200pF for the data bus and 100pF for address and control lines

D. Although static by design, testing guarantees $t_w(\Phi H)$ of 200 μsec maximum.

A.C. TIMING DIAGRAM

Timing measurements are made at the following voltages, unless otherwise specified:

	"1"	"0"
CLOCK	V_{cc}–.6	.8V
OUTPUT	2.0 V	.8V
INPUT	2.0 V	.8V
FLOAT	ΔV	±0.5 V



12.0 Z80 INSTRUCTION BREAKDOWN BY MACHINE CYCLE

This section tabulates each Z80 instruction type and breaks each instruction down into its machine cycles and corresponding T States. The different standard machine cycles (OP Code Fetch, Memory Read, Port Read, etc.) are described in Section 4.0 of this manual. This chart will allow the system designer to predict what the Z80 will do on each clock cycle during the execution of a given instruction. The instruction types are listed together by functions and in the same order as the Tables in Section 7.

The best way to learn how to use these tables is to look at a few examples. The first example is to register exchange instructions (LD r, s) where r,s can be any of the following CPU Registers: B,C,D,E,H,L, or A. The instruction breakdown table shows this instruction to have one machine cycle (M1) four T-States long (number in parenthesis) which is an OP Code Fetch. Referring to Figure 4.0-1 one sees the standard form for an OP Code Fetch and the state of the CPU bus during these four T-States. Taking the next instruction shown (LD r, n) which loads one of the previous registers with data or immediate value "n" one finds the breakdown to be a four T-State OP Code Fetch followed by a three T-State Operand Data Read. An Operand Data Read takes the form of the Standard Memory Read shown in Figure 4.0-2.

After these two simple examples, a more complex one is in order. The LD r, (IX+d) is the first double byte OP Code shown and executes as follows: First there are two M1 cycles (and related memory refreshes) followed by an Operand Data Read of the displacement "d". Next M3 consists of a five T-State Internal Operation which is the calculation of the Indexed address (IX+d). The last machine cycle (M4) consists of a Memory Read of the data continued in address IX+d and the loading of register "r" with that data.

The LD dd, (nn) instruction loads an internal 16-bit register pair with the contents of the memory location specified in the Operand Bytes of the instruction. This instruction is four bytes long (two bytes of OP Code + two bytes of Operand Address). As shown, there are two M1 cycles to fetch the OP Code and then two Machine Cycles to read the Operand Addresses, low order byte first. Machine cycle 4 is a read of memory to obtain the data for the low order register (e.g., C of BC, E of DE and L of HL) followed by a read of the data for the high order register.

The first instruction to use the Stack Register is the PUSH qq instruction which executes as follows: Machine cycle 1 is extended by one cycle and the Stack Pointer is decremented in the extra T-State to point to an empty location on the Stack. Machine cycle 2 is a write of the high byte of the referenced register to the address contained in the Stack Pointer. The Stack Pointer is again decremented and a write of the low byte of the referenced register is made to the Stack in Machine Cycle 3. Note that the Stack Pointer is left pointing to the last data referenced on the Stack. The block transfer instructions such as LDI and LDIR are very similar. LDI is 16 T-States long and is composed of a double byte OP Code Fetch (two memory refreshes) followed by a memory read and a memory write. The memory write is 5 T-States long to allow updating of the block length counter —BC. The repetitive form of this instruction (LDIR) has an additional Machine Cycle (M4) of 5 T-States to allow decrementing of the Program Counter by two (PC-2) which results in refetching of the OP Code (LDIR). Each movement of data by this instruction is 21 T-States long (except the last) and the refetching of the OP Codes results in memory refresh occurring as well as the sampling of interrupts and $\overline{\text{BUSRQ}}$.

The $\overline{\text{NMI}}$ Interrupt sequence is 11 T-States long with the first M1 being a dummy OP Code Fetch of 5 T-States long. The Program Counter is not advanced, the OP Code on the data bus is ignored and an internal Restart is done to address 66H. The following two Machine Cycles are a write of the Program Counter to the Stack.

The $\overline{\text{INT}}$ Mode 0 is the 8080A mode and requires the user to place an instruction on the data bus for the CPU to execute. If a RST instruction is used, the CPU stacks the Program Counter and begins execution at the Restart Address. If a CALL instruction is used, the CALL Op Code is placed on the data bus during the INTA cycle (M1). M2 and M3 are

normal Memory Read cycles (not INTA cycles) of the CALL addresses (low byte first). Program Counter is stacked in M4 and M5.

Mode 2 is used by the Z80 System Peripherals and operates as follows: During the INTA cycle (M1) a Vector is sent in from the highest priority interrupting device. M2 and M3 are used to Stack the Program Counter. The Vector (low byte) and an internal Interrupt Register (I) from a pointer to a table containing the addresses of Interrupt Service Routines. During M4 and M5 the Service Routines address is read from this table into the CPU. The next M1 cycle will fetch an OP Code from the address received is M4 and M5.

LEGEND

IO	— Internal CPU Operation	ODL	— Operand Data Read of Low Byte
MR	— Memory Read	PR	— Port Read
MRH	— Memory Read of High Byte	PW	— Port Write
MRL	— Memory Read of Low Byte	SRH	— Stack Read of High Byte
MW	— Memory Write	SRL	— Stack Read of Low Byte
MWH	— Memory Write of High Byte	SWH	— Stack Write of High Byte
MWL	— Memory Write of Low Byte	SWL	— Stack Write of Low Byte
OCF	— Op Code Fetch	( )	— Number of T-States in that Machine Cycle
ODH	— Operand Data Read of High Byte		

Z80 INSTRUCTION BREAKDOWN BY MACHINE CODE

INSTRUCTION TYPE	BYTES	MACHINE CYCLE M1	M2	M3	M4	M5
LD r, s	1	OCF (4)				
LD r, n	2	OCF (4)	OD (3)			
LD r, (HL) LD (HL), r	1	OCF (4) OCF (4)	MR (3) MW (3)			
LD r, (IX+d) LD (IX+d), r	3	OCF (4)/OCF (4) OCF (4)/OCF (4)	OD (3) OD (3)	IO (5) IO (5)	MR (3) MW (3)	
LD (HL), n	2	OCF (4)	OD (3)	MW (3)		
LD A, $\binom{BC}{DE}$	1	OCF (4)	MR (3)			
LD $\binom{BC}{DE}$, A		OCF (4)	MW (3)			
LD A, (nn) LD (nn), A	3	OCF (4) OCF (4)	ODL (3) ODL (3)	ODH (3) ODH (3)	MR (3) MW (3)	
LD A, $\binom{I}{R}$ LD $\binom{I}{R}$, A	2	OCF (4)/OCF(5)				
LD dd, nn	3	OCF (4)	ODL (3)	ODH (3)		
LD IX, nn	4	OCF (4)/OCF (4)	ODL (3)	ODH (3)		
LD HL, (nn) LD (nn), HL	3	OCF (4) OCF (4)	ODL (3) ODL (3)	ODH (3) ODH (3)	MRL (3) MWL (3)	MRH (3) MWH (3)

LD dd, (nn)
LD (nn), dd
LD IX, (nn)
LD (nn), IX
	4	OCF (4)/OCF (4)
		OCF (4)/OCF (4)
		OCF (4)/OCF (4)
		OCF (4)/OCF (4)
			ODL (3)
			ODL (3)
			ODL (3)
			ODL (3)
				ODH (3)
				ODH (3)
				ODH (3)
				ODH (3)
					MRL (3)
					MWL (3)
					MRL (3)
					MWL (3)
						MRH (3)
						MWH (3)
						MRH (3)
						MWH (3)

LD SP, HL	1	OCF (6)				
LD SP, IX	2	OCF (4)/OCF (6)				
PUSH qq	1	OCF (5) SP-1 →	SWH (3) SP-1 →	SWL (3)		
PUSH IX	2	OCF (4)/OCF (5) SP-1 →	SWH (3) SP-1 →	SWL (3)		
POP qq	1	OCF (4)	SRH (3) SP+1 →	SRL (3)	SP+1 →	
POP IX	2	OCF (4)/OCF (4)	SRH (3) SP+1 →	SRL (3)	SP+1 →	
EX DE, HL	1	OCF (4)				
EX AF, AF'	1	OCF (4)				

		MACHINE CYCLE				
INSTRUCTION TYPE	**BYTES**	**M1**	**M2**	**M3**	**M4**	**M5**
EXX	1	OCF (4)				
EX (SP), HL	1	OCF (4)	SRL (3) SP+1 →	SRH (4)	SWH (3) SP-1 →	SWL (5)
EX (SP), IX	2	OCF (4)/OCF (4)	SRL (3) SP+1 →	SRH (4)	SWH (3) SP-1 →	SWL (5)
LDI LDD CPI CPD	2	OCF (4)/OCF (4)	MR (3)	MW (5)		
LDIR LDDR CPIR CPDR	2	OCF (4)/OCF (4)	MR (3)	MW (5)	IO (5)* *only if BC ≠ 0	
ALU A, r ADD ADC SUB SBC AND OR XOR CP	1	OCF (4)				
ALU A, n	2	OCF (4)	OD (3)			
ALU A, (HL)	1	OCF (4)	MR (3)			
ALU A, (IX+d)	3	OCF (4)/OCF (4)	OD (3)	IO (5)	MR (3)	
DEC INC r	1	OCF (4)				
DEC INC (HL)	1	OCF (4)	MR (4)	MW (3)		
DEC INC (IX+D)	2	OCF (4)/OCF (4)	OD (3)	IO (5)	MR (4)	MW (3)
DAA CPL CCF SCF NOP HALT DI EI	1	OCF (4)				
NEG IMO IM1 IM2	2	OCF (4)/OCF (4)				

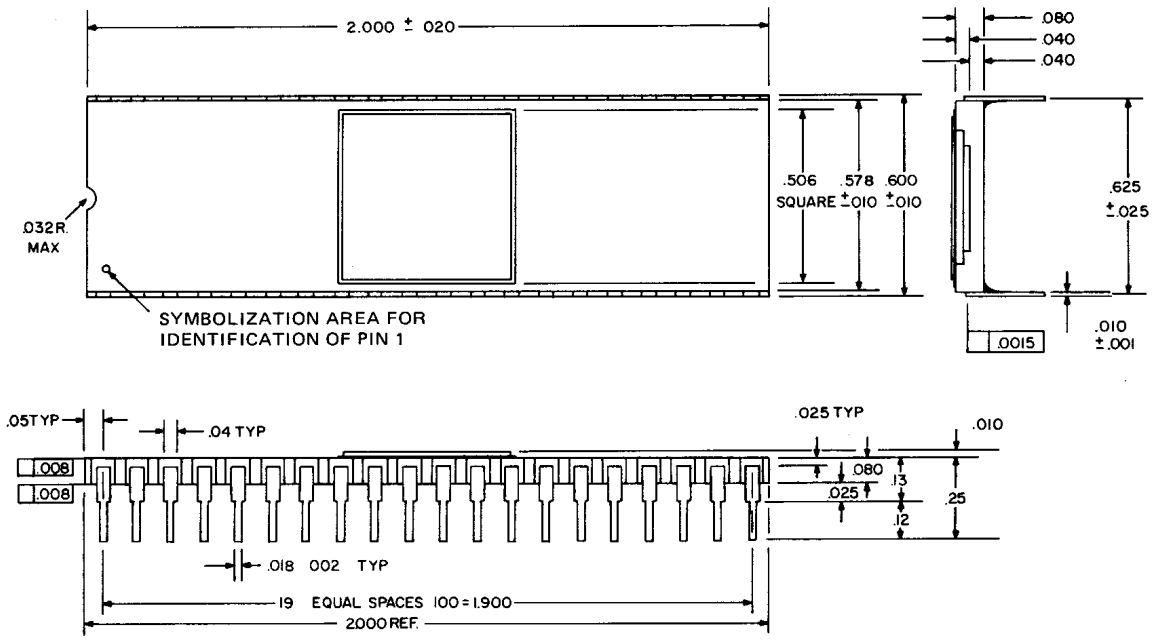
INSTRUCTION TYPE	BYTES	MACHINE CYCLE M1	M2	M3	M4	M5
ADD HL, ss	1	OCF (4)	IO (4)	IO (3)		
ADC HL, ss SBC HL, ss ADD IX, pp	2	OCF (4)/OCF (4)	IO (4)	IO (3)		
INC ss DEC ss	1	OCF (6)				
DEC IX INC IX	2	OCF (4)/OCF (6)				
RLCA RLA RRCA RRA	1	OCF (4)				
RLC r RL RRC RR SLA SRA SRL	2	OCF (4)/OCF (4)				
RLC (HL) RL RRC RR SLA SRA SRL	2	OCF (4)/OCF (4)	MR (4)	MW (3)		
RLC (IX+d) RL RRC RR SLA SRA SRL	4	OCF (4)/OCF (4)	OD (3)	IO (5)	MR (4)	MW (3)
RLD RRD	2	OCF (4)/OCF (4)	MR (3)	IO (4)	MW (3)	
BIT b, r SET RES	2	OCF (4)/OCF (4)				

		MACHINE CYCLE				
INSTRUCTION TYPE	**BYTES**	**M1**	**M2**	**M3**	**M4**	**M5**
BIT b, (HL)	2	OCF (4)/OCF (4)	MR (4)			
SET b, (HL) RES	2	OCF (4)/OCF (4)	MR (4)	MW (3)		
BIT b, (IX+d)	4	OCF (4)/OCF (4)	OD (3)	IO (5)	MR (4)	
SET b, (IX+d) RES	4	OCF (4)/OCF (4)	OD (3)	IO (5)	MR (4)	MW (3)
JP nn JP cc, nn	3	OCF (4)	ODL (3)	ODH (3)		
JR e	2	OCF (4)	OD (3)	IO (5)		
JR C, e JR NC, e JR Z, e JR NZ, e	2	OCF (4)	OD (3)	IO (5)* * If condition is met		
JP (HL)	1	OCF (4)				
JP (IX)	2	OCF (4)/OCF (4)				
DJNZ, e	2	OCF (5)	OD (3)	IO (5)* * If B≠ 0		
CALL nn CALL cc, nn cc true	3	OCF (4)	ODL (3)	ODH (4) SP-1 →	SWH (3) SP-1 →	SWL (3)
CALL cc, nn cc false	3	OCF (4)	ODL (3)	ODH (3)		
RET	1	OCF (4)	SRL (3) SP+1 →	SRH (3) →	 SP+1 →	
RET cc	1	OCF (5)	SRL (3)* * If cc is true SP+1 →	SRH (3)* →	 SP+1 →	
RETI RETN	2	OCF (4)/OCF (4)	SRL (3) SP+1 →	SRH (3) →	 SP+1 →	
RST p	1	OCF (5) SP-1 →	SWH (3) SP-1 →	SWL (3) →		

INSTRUCTION TYPE	BYTES	MACHINE CYCLE M1	M2	M3	M4	M5
IN A, (n)	2	OCF (4)	OD (3)	PR (4)		
IN r, (c)	2	OCF (4)/OCF (4)	PR (4)			
INI IND	2	OCF (4)/OCF (5)	PR (4)	MW (3)		
INIR INDR	2	OCF (4)/OCF (5)	PR (4)	MW (3)	IO (5)	
OUT (n) , A	2	OCF (4)	OD (3)	PW (4)		
OUT (C), r	2	OCF (4)/OCF (4)	PW (4)			
OUTI OUTD	2	OCF (4)/OCF (5)	MR (3)	PW (4)		
OTIR OTDR	2	OCF (4)/OCF (5)	MR (3)	PW (4)	IO (5)	
INTERRUPTS						
NMI	–	OCF (5) * SP-1 →	SWH (3) SP-1 →	SWL (3)	*Op Code Ignored	
INT						
MODE 0	–	INTA (6) (CALL INSERTED)	ODL (3)	ODH (4) SP-1 →	SWH (3) SP-1 →	SWL (3)
	–	INTA (6) (RST INSERTED) SP-1 →	SWH (3) SP-1 →	SWL (3)		
MODE 1		INTA (7) (RST 38H INTERNAL) SP-1 →	SWH (3) SP-1 →	SWL (3)		
MODE 2	–	INTA (7) (VECTOR SUPPLIED) SP-1 →	SWH (3) SP-1 →	SWL (3)	MRL (3)	MRH (3)

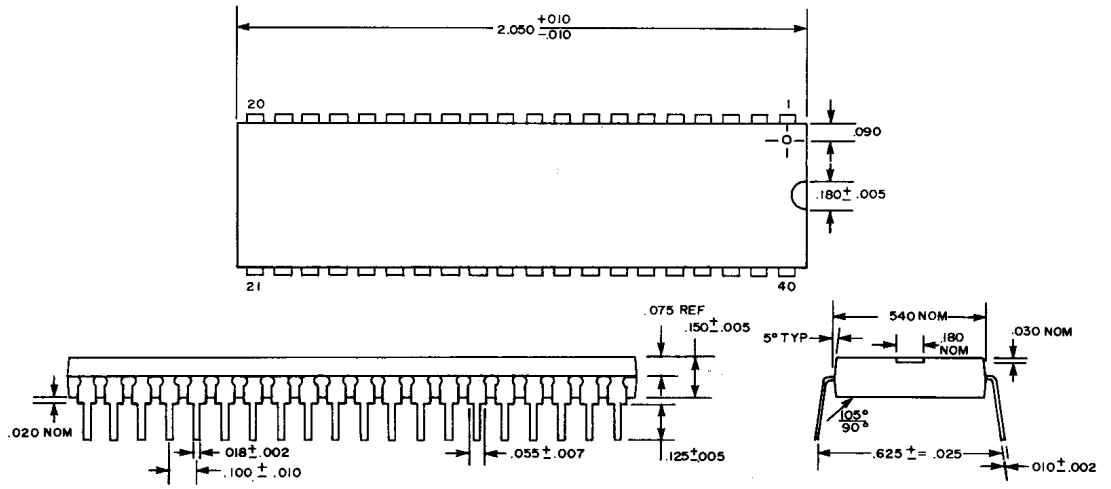
13.0 PACKAGE DESCRIPTION AND ORDERING INFORMATION

PACKAGE DESCRIPTION – 40 Pin Dual-In-Line Ceramic Package



Family

PACKAGE DESCRIPTION – 40-Pin Dual-In-Line Plastic Package



ORDERING INFORMATION

PART NO.	PACKAGE TYPE	MAX CLOCK FREQUENCY	TEMPERATURE RANGE
MK3880N Z80-CPU	Plastic	2.5 MHz	0° to + 70° C
MK3880P Z80-CPU	Ceramic	2.5 MHz	
MK3880N-4 Z80-CPU	Plastic	4.0 MHz	
MK3880P-4 Z80A-CPU	Ceramic	4.0 MHz	
MK3880P-10 Z80-CPU	Ceramic	2.5 MHz	-40°C to +85°C
MK3880P-20 Z80-CPU	Ceramic	2.5 MHz	-55° C to +125° C